

UNCLASSIFIED

AD NUMBER

ADB381215

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; MAY 2012. Other requests shall be referred to Air Force Research Laboratory, ATTN: AFRL/RWC, Wright-Patterson AFB, OH 45433-7334. This document contains export-controlled technical data.

AUTHORITY

AFRL memo dtd 21 Mar 2013

THIS PAGE IS UNCLASSIFIED



AFRL-RY-WP-TR-2012-0111, V2



EMBEDDED INFORMATION SYSTEMS TECHNOLOGY SUPPORT (EISTS)

Task Order 0006: Vulnerability Path Analysis and Demonstration (VPAD) Volume 2 –White Box Definitions of Software Fault Patterns

Ben A. Calloni, Ph.D., P.E., Djenana Campara, and Nikolai Mansourov, Ph.D.

Lockheed Martin Corporation and KDM Analytics Inc.

DECEMBER 2011

Final Report

Distribution authorized to U.S. Government Agencies and their contractors; Critical Technology; May 2012. Other requests for this document shall be referred to AFRL/RWYC, Wright-Patterson Air Force Base, OH 45433-7334.

WARNING – This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec. 2751, et seq.) or the Export Administration Act of 1979, as amended, Title 50, U.S.C., App. 2401, et seq. Violations of these export laws are subject to severe criminal penalties. Disseminate in accordance with the provisions of DoD Directive 5230.25. (Include this statement with any reproduced portions.)

DESTRUCTION NOTICE – Destroy by any method that will prevent disclosure of contents or reconstruction of the document.

See additional restrictions described on inside pages

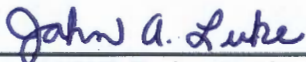
**AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE**

NOTICE AND SIGNATURE PAGE

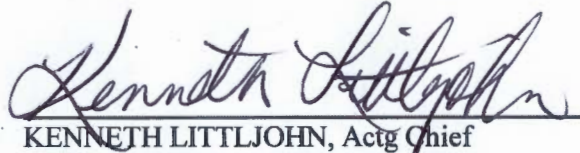
Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

Qualified requestors may obtain copies of this report from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

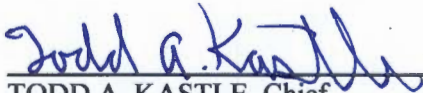
AFRL-RY-WP-TR-2012-0111, V2 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.



JAHN A. LUKE, Project Engineer
Distributed Collaborative Sensor System
Technology Branch
Integrated Electronic & Net-Centric
Warfare Division



KENNETH LITTLEJOHN, Actg Chief
Distributed Collaborative Sensor System
Technology Branch
Integrated Electronic & Net-Centric
Warfare Division



TODD A. KASTLE, Chief
Integrated Electronic and Net-Centric
Warfare Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>						
1. REPORT DATE (DD-MM-YY) December 2011		2. REPORT TYPE Final		3. DATES COVERED (From - To) 31 March 2009 – 30 November 2011		
4. TITLE AND SUBTITLE EMBEDDED INFORMATION SYSTEMS TECHNOLOGY SUPPORT (EISTS) Delivery Order 0006: Vulnerability Path Analysis and Demonstration (VPAD) Volume 2 – White Box Definitions of Software Fault Patterns					5a. CONTRACT NUMBER F33615-02-D-4035-0006	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER 78612F	
6. AUTHOR(S) Ben A. Calloni, Ph.D., P.E., Djenana Campara, and Nikolai Mansourov, Ph.D.					5d. PROJECT NUMBER 4035	
					5e. TASK NUMBER 00	
					5f. WORK UNIT NUMBER 40350006	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lockheed Martin Corporation P.O. Box 748 Fort Worth, TX 76101					8. PERFORMING ORGANIZATION REPORT NUMBER KDM Analytics Inc. 3730 Richmond Rd, Suite 204 Ottawa, ON K2H 5B9	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command United States Air Force					10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/RYWC	
					11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2012-0111, V2	
12. DISTRIBUTION/AVAILABILITY STATEMENT Distribution authorized to U.S. Government Agencies and their contractors; Critical Technology; May 2012. Refer other requests for this document to AFRL/RYWC, Wright-Patterson Air Force Base, OH 45433-7320. This report is the second of two volumes.						
13. SUPPLEMENTARY NOTES Export control restrictions apply. Report contains color.						
14. ABSTRACT AFRL's Embedded Information Systems Technology Support (EISTS) contract vehicle was used to support the Vulnerability Path Analysis and Demonstration (VPAD) project sponsored by the Office of the Assistant Secretary of Defense (OASD) for Network and Information Integration (NII) - Department of Defense (DoD) Chief Information Officer (CIO), supporting the Globalization Task Force (Information Assurance). In this effort, LM Aero and KDM Analytics were tasked to support OASD in providing continued research in the area of Software Assurance (SwA) and to further work toward the development of SwA Ecosystem based on Object Management Group (OMG) standards. Focus of this effort was to advance semantic formalisms of Software Fault Patterns (weaknesses) and to create a Test Case Generator (TCG) capable of automatically generate various programming language test cases of fault code constructs. Such constructs could then serve as test cases to test the effectiveness of various static code analysis tools, thus providing enhanced tooling to reduce software vulnerabilities. This volume focuses on the Software Fault Pattern work performed by KDM Analytics.						
15. SUBJECT TERMS OMG SwA Ecosystem, software assurance, software vulnerabilities, software fault patterns, static analysis, common weakness enumeration, CWE formalization, vulnerability path analysis, code complexity taxonomy						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 176	19a. NAME OF RESPONSIBLE PERSON (Monitor) Jahn A. Luke	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include Area Code) (937) 528-8033	

Table of Contents

Section	Page
List of Figures	iv
List of Tables	iv
Section 1. Summary	1
1.1 Statement of Work.....	1
1.1.1 Naming Change from WK to SFP	2
1.2 Introduction	2
1.3 The Systems Context.....	3
1.4 Software Fault Pattern	4
1.5 Discernible and Non-Discernable Characteristics.....	9
Section 2. Obtaining White Box Definitions	11
2.1 Bottom-Up: Cluster Discovery Process	11
2.2 Top-Down: Software Fault Pattern Discovery Process.....	12
2.2.1 Parameterization of Software Fault Patterns.....	13
2.3 A Quick Summary of Software Fault Pattern Extraction Process.....	15
Section 3. Summary of Clusters and Software Fault Patterns	17
Section 4. Clusters and Software Fault Patterns	22
4.1 Primary Cluster: Risky Values	22
4.1.1 Secondary Cluster: Glitch in Computation	23
4.2 Primary Cluster: Unused Entities	29
4.2.1 Secondary Cluster: Unused Entities.....	30
4.3 Primary Cluster: API.....	31
4.3.1 Secondary Cluster: Use of an Improper API	31
4.4 Primary Cluster: Exception Management	34
4.4.1 Secondary Cluster: Unchecked Status Condition	35
4.4.2 Secondary Cluster: Ambiguous Exception Type	38
4.4.3 Secondary Cluster: Incorrect exception Behavior	39
4.5 Primary Cluster: Memory Access	40
4.5.1 Secondary Cluster: Faulty Pointer Use	41
4.5.2 Secondary Cluster: Faulty Buffer Access	42
4.5.3 Secondary Cluster: Faulty String Expansion	45
4.5.4 Secondary Cluster: Incorrect Buffer Length Computation	45
4.5.5 Secondary Cluster: Improper NULL Termination.....	46
4.6 Primary Cluster: Memory Management.....	47
4.6.1 Secondary Cluster: Faulty Memory Release.....	48
4.7 Primary Cluster: Resource Management.....	49
4.7.1 Secondary Cluster: Unrestricted Consumption.....	50
4.7.2 Secondary Cluster: Failure to release resource	51
4.7.3 Secondary Cluster: Faulty Resource Use.....	52
4.7.4 Secondary Cluster: Life Cycle	53
4.8 Primary Cluster: Path Resolution	53
4.8.1 Secondary Cluster: Path Traversal.....	54
4.8.2 Secondary Cluster: Failed Chroot Jail	61
4.8.3 Secondary Cluster: Link in Resource Name Resolution.....	62

4.9	Primary Cluster: Synchronization	64
4.9.1	Secondary Cluster: Missing Lock	65
4.9.2	Secondary Cluster: Race Condition Window	67
4.9.3	Secondary Cluster: Multiple Locks/Unlocks	69
4.9.4	Secondary Cluster: Unrestricted Lock	70
4.10	Primary Cluster: Information Leak	71
4.10.1	Secondary Cluster Exposed Data	72
4.10.2	Secondary Cluster: State Disclosure	83
4.10.3	Secondary Cluster: Exposure Through Temporary files	84
4.10.4	Secondary Cluster: Other Exposures	85
4.10.5	Secondary Cluster: Insecure Session Management	86
4.11	Primary Cluster: Tainted Input	86
4.11.1	Secondary Cluster: Tainted Input to Command	88
4.11.2	Secondary Cluster: Tainted Input to Variable	103
4.11.3	Secondary Cluster: Composite Tainted Input	104
4.11.4	Secondary Cluster: Faulty input Transformation	105
4.11.5	Secondary Cluster: Incorrect Input Handling	106
4.11.6	Secondary Cluster: Tainted Input to Environment	108
4.12	Primary Cluster: Entry Points	110
4.12.1	Secondary Cluster: Unexpected Access Points	110
4.13	Primary Cluster: Authentication	112
4.13.1	Secondary Cluster: Authentication Bypass	113
4.13.2	Secondary Cluster: Faulty Endpoint Authentication	115
4.13.3	Secondary Cluster: Missing Endpoint Authentication	116
4.13.4	Secondary Cluster: Digital Certificate	117
4.13.5	Secondary Cluster: Missing Authentication	118
4.13.6	Secondary Cluster: Insecure Authentication Policy	119
4.13.7	Secondary Cluster: Multiple binds to the Same Port	119
4.13.8	Secondary Cluster: Hardcoded Sensitive Data	120
4.13.9	Secondary Cluster: Unrestricted Authentication	121
4.14	Primary Cluster: Access Control	122
4.14.1	Secondary Cluster: Insecure Resource Access	123
4.14.2	Secondary Cluster: Insecure Resource Permissions	124
4.14.3	Secondary Cluster: Access Management	125
4.15	Primary Cluster: Privilege	126
4.15.1	Secondary Cluster: Privilege	126
4.16	Primary Cluster: Channel	128
4.16.1	Secondary Cluster: Channel Attack	129
4.16.2	Secondary Cluster: Protocol Error	130
4.17	Primary Cluster: Cryptography	130
4.17.1	Secondary Cluster: Broken Cryptography	131
4.17.2	Secondary Cluster: Weak Cryptography	132
4.18	Primary Cluster: Malware	132
4.18.1	Secondary Cluster: Malicious Code	133
4.18.2	Secondary Cluster: Covert Channel	134
4.19	Primary Cluster: Predictability	134

4.19.1	Secondary Cluster: Predictability	135
4.20	Primary Cluster: UI	136
4.20.1	Secondary Cluster: Feature	137
4.20.2	Secondary Cluster: Information Loss	137
4.20.3	Secondary Cluster: Security.....	138
4.21	Primary Cluster: Other	138
4.21.1	Secondary Cluster: Architecture	139
4.21.2	Secondary Cluster: Design.....	140
4.21.3	Secondary Cluster: Implementation.....	143
4.21.4	Secondary Cluster: Compiler.....	144
APPENDIX A: Software Fault Patterns		145
APPENDIX B: Software Fault Patterns and Corresponding Impacts		164
List of Acronyms, Abbreviations, and Symbols		168

List of Figures

Figure	Page
Figure 1. Weakness Conceptual Model	6
Figure 2. Weakness Logical Model	7
Figure 3. Weakness Definition Separated From the Corresponding Computation	8
Figure 4. Classification of Impacts	9
Figure 5. SFP Parameterization Process	14

List of Tables

Table	Page
Table 1. Software Fault Pattern Extraction Process Steps	15
Table 2. Software Fault Clusters Summary Table	18
Table 3. Input Commands of TIC Type	89
Table 4. Destination Commands of TIC Type	89
Table 5. Special Characters of TIC Type	89
Table 6. CWEs in Relationship to Parameters of TIC Type	91
Table 7. Example: Concrete Parameters for Parameterized TIC SFP	93
Table 8. Software Fault Patterns	145
Table 9. Software Fault Patterns with Corresponding Impacts	164

Section 1. Summary

This effort is the result of Software Assurance (SwA) research performed by KDM Analytics Inc as a supplier to Lockheed Martin Aeronautics Company under Contract F33615-02-D-4035/0006, entitled “Vulnerability Path Analysis and Demonstration (VPAD)” for the Office of the Assistant Secretary of Defense (OASD) for Network and Information Integration (NII) - Department of Defense (DoD) Chief Information Officer (CIO), Globalization Task Force (Information Assurance). This document provides a catalog of White Box Definitions with defined parameters for Software Fault Patterns (SFPs). The VPAD program is managed by the Air Force Research Laboratory (AFRL).

This document provides an overview of the process leading to the systematic classification of software weaknesses that enables automation – the Software Fault Pattern (SFP) approach, describes linkages to Common Weakness Enumerations (CWEs), and the white box definitions for the parameterized Software Fault Patterns.

The SFP work for this project was divided in two phases: (a) Phase I which performed the work outlined in contract Statement of Work (SOW) Requirements 3.1.1 and 3.1.3, and (b) Phase 2 which performed the work as per SOW Requirement 3.1.5. The final results, as the outcome of both phases are captured in this document and an accompanying Excel spreadsheet document entitled “Phase II SFP Deliverable Final”, which is available upon request.

1.1 Statement of Work

The following requirements were captured from the Supplier SOW from which the work was performed:

“3.1.1 Identify and Develop White Box Definitions for Weakness Kernels”.

The Supplier shall identify and develop white box definitions for Weakness Kernels (WKs), particularly those that can be used to define Common Weakness Enumerations (CWE). In most cases, the identified WKs will be common to more than one CWE. Once completed, these WK white box definitions will be ready for the formalization process and at that point could be integrated into a standards-based tool analysis approach, benefiting both real-time embedded and enterprise software assurance systems. The Supplier shall document the analysis results and definitions in a technical report.

“3.1.3 Link the Weakness Kernels to their Corresponding CWEs”.

The Supplier shall identify the CWEs that employ the identified WKs, adjusting the WK definitions as required.”

“3.1.5 Parameterize the remaining 31 Software Fault Patterns and Update the Previous 19 Software Fault Patterns Documentation”.

During the process of extracting SFPs from CWE clusters, the Supplier (KDM Analytics) noticed a specific SFP characteristic which when parameterized properly could lead to

the SFP becoming a Compliance Point. KDM Analytics parameterized 19 SFPs in a previous effort. The same parameterization analysis will be performed across 31 remaining SFPs, and the earlier 19 will be updated as required.

1.1.1 Naming Change from WK to SFP

During the final Phase I review on 29 Sep 2009, it was agreed that the term “Software Fault Patterns” would be used instead of “Weakness Kernel” which was referenced in the SOW.

1.2 Introduction

One of the first steps in preventing cyber attacks is to collect and efficiently manage knowledge about exploitable weaknesses in such a way that can be utilized by a community to build more comprehensive prevention, detection and mitigation solutions.

Several classifications of vulnerabilities have been provided by the community; however it has been observed that all existing classifications *resist automation*. The primary objective of this document is to bring clarity and precision to the study of vulnerabilities by describing a systematic catalog of weaknesses that enables automation – the Software Fault Patterns.

The first step in this direction has been done: the input to the VPAD program is the knowledge of the software weaknesses that is available as the Common Weakness Enumeration (CWE) taxonomy. The VPAD program further organizes this knowledge and makes it formal in order to deliver executable specifications for each exploitable weakness, making the knowledge consistent and measurable. This is accomplished through several transformations of knowledge captured informally as the CWE taxonomy.

1. The first transformation normalizes weakness definitions in the form of patterns and the corresponding pattern rules associated with white-box knowledge.
2. The second transformation extracts common patterns and associated conditions (pattern rules) collectively referred to as Software Fault Patterns and redefines existing weaknesses as specializations of the common patterns.
3. The third transformation is to redefine each Software Fault Pattern with focus on invariant core and variation points as parameters. This way, weakness specializations are represented by parameters in the corresponding Software Fault Pattern.
4. The fourth transformation formalizes each Software Fault Pattern (SFP) using Semantics of Business Vocabulary and Business Rules (SBVR) and Knowledge

Discovery Metamodel (KDM), resulting in Structured English and XMI representations of each weakness.

5. The final transformation is the conversion of each XMI into an executable Prolog rule.

VPAD Phases I & II of the SFP work focused on transformations described above under steps 1, 2 and 3. Due to limited funds, instead of performing steps 1 and 2 in the logical order (which gives a better guarantee for the quality and comprehensiveness of the common patterns and rules, resulting in natural clusters of weaknesses), a more automated approach was used which constructed an approximated solution. The approach to these first 2 steps of knowledge transformations is to perform step 2 first by using automated semantic clustering technique to perform groupings of CWEs based on their descriptions, and then performing manual examination of the resulting clusters, and then subsequently extracting the SFPs. The reduction of the effort results from shifting the manual effort from inspecting and normalizing individual CWEs to analyzing a much smaller number of semantic clusters constructed automatically (some additional effort was spent in managing the automated clustering process). Although this approximation gave us large savings in time and effort, it is important to keep in mind that some adjustments of the existing CWE definitions are required. The loss of quality in CWE classification results from the fact that the CWE descriptions and other characteristics are informal. This often leads to overlaps between the classes of faulty computations implied by each CWE element. In order to develop a more systematic and automatable classification, we re-examined and re-grouped the individual computations described by the CWE elements while maintaining a uniform viewpoint – the patterns of the system artifact that are indicators of the each computation.

1.3 The Systems Context

In general, a *computation* is a sequence of steps/events performed by the system or one of the activities within the system. The computation is performed by the *code*, which is supported by other components of the system. Code provides the *constraints* to computations and therefore determines what computations can occur.

From the systems perspective, a system is a collection of activities that exchange information to achieve some common purpose. Computations occur at *system nodes* that are connected by *channels*. Following the NIST CVSS approach, we distinguish *local* channels between system nodes deployed at the same machine; *adjacent network* channels between system nodes deployed at the same local area network; and *remote* channels. This distinction is important because it determines the class of *access* required in order to exploit vulnerability. Each system node performs computation to provide *services* to other system nodes or the environment of the entire system. Data interchanges use channels. We distinguish between *data at rest* (for example, databases), *data in motion* (data in the channel) and *data in use* (data flowing inside the computation).

Certain steps are specific to the given system. However, there are certain sequences of steps that are common to large families of systems. For example, such common sequences are related to input processing, authentication, access control, cryptography, information output, resource management, memory buffer management, exception management. The catalog of faulty computations should focus at the computation that is common to large families of systems; however it should scale well to enable customization for a specific targeted assessment.

Vulnerability is defined as “a bug, flaw, weakness, or exposure of an application, system, device, or service that could lead to a failure of confidentiality, integrity, or availability” [source: NIST Interagency Report 7435, August 2007]. In other words, “vulnerability” is a computation that can be exploited to produce (negative) *impact*. Certain computations in the system are designed to mitigate the vulnerabilities. These computations and the corresponding mechanisms and “places” in the code are called “safeguards”. A “*faulty computation*” is defined as either a computation that has direct impact or a computation corresponds to an incorrect *safeguard*.

A “*computation foot-hold*” is a known construct, an entry point, or an API call manifested in the system’s artifacts, such that it is a necessary condition for the computation. Certain places in the code can be utilized to directly cause (negative) impact. Such places are foot-holds for the impacting computation sequences. Safeguards (as computation sequences) also have foot-holds, related to the safeguard itself as well as to the protected region. Software fault patterns are elements of the catalog of the unique places in the code associated with faulty computations that directly have impact or to the failed safeguards.

The Software fault pattern approach provides a catalog of the faulty computations, and focuses on the “places” in the code, where there are *indicators* of particular computations. In particular, the Software Fault Pattern approach focuses at the “*foot-holds*” which are places in the code where there are the necessary conditions for vulnerabilities. Therefore the Software Fault Pattern approach is driven by the patterns in the code as they determine particular classes of faulty computations. This viewpoint is constructive and systematic and therefore enables automation. Uniform viewpoint makes the Software Fault Pattern approach systematic and repeatable.

1.4 Software Fault Pattern

The Software Fault Pattern (or SFP) is a common pattern with one or more associated pattern rules (conditions) representing a family of faulty computations. The generalized SFP definition refers to the entire secondary cluster and is arranged into invariant core and variation points. To ensure full coverage, variation points are identified through top-down view of entire cluster space. Once all variation points are identified they are mapped to specific parameters. In other words, parameters introduce additional details for the generalized definition, focusing at the variation points.

SFPs map to multiple CWEs in such a way that each CWE in the family can be defined as a specialization of the SFP where specialization is defined by one or more SFP parameters. Thus, CWEs serve as a reporting mechanism.

Identified Software Fault Pattern definitions will lead to the development of more accurate testing tools and also improve developer education since it is easier to manage the knowledge of fewer SFPs than hundreds of CWEs. They also provide for a more cost effective formalization, since for each CWE, only the extension to a formalized SFP is required.

In order to express SFPs as patterns and associated pattern rules that enable grouping by specialization in a consistent, measurable and comparable way, we developed conceptual and logical models that focus on essential characteristics expressed as:

- elements
- relations between elements and
- rules describing relations

In accordance with the principles of the Model-Driven Architecture, the logical model is the key artifact, where a multitude of physical models can be automatically derived from it for selected implementation technologies.

A conceptual model of a software fault pattern is presented in Figure 1. The conceptual model facilitates readable definitions of SFPs in structured English known as contractual formalization. It removes any ambiguity from the SFP and separates white box discernible from non discernible properties, although both could be retained in the SFP definition. Each pattern has a start and an end statement connected by a path constrained by particular conditions. The start statement determines the source of data while the end statement determines the data sink and the weakness corresponds to any computation between the start and end statements that propagates certain data properties that satisfy a certain end-to-end data condition associated with the computation.

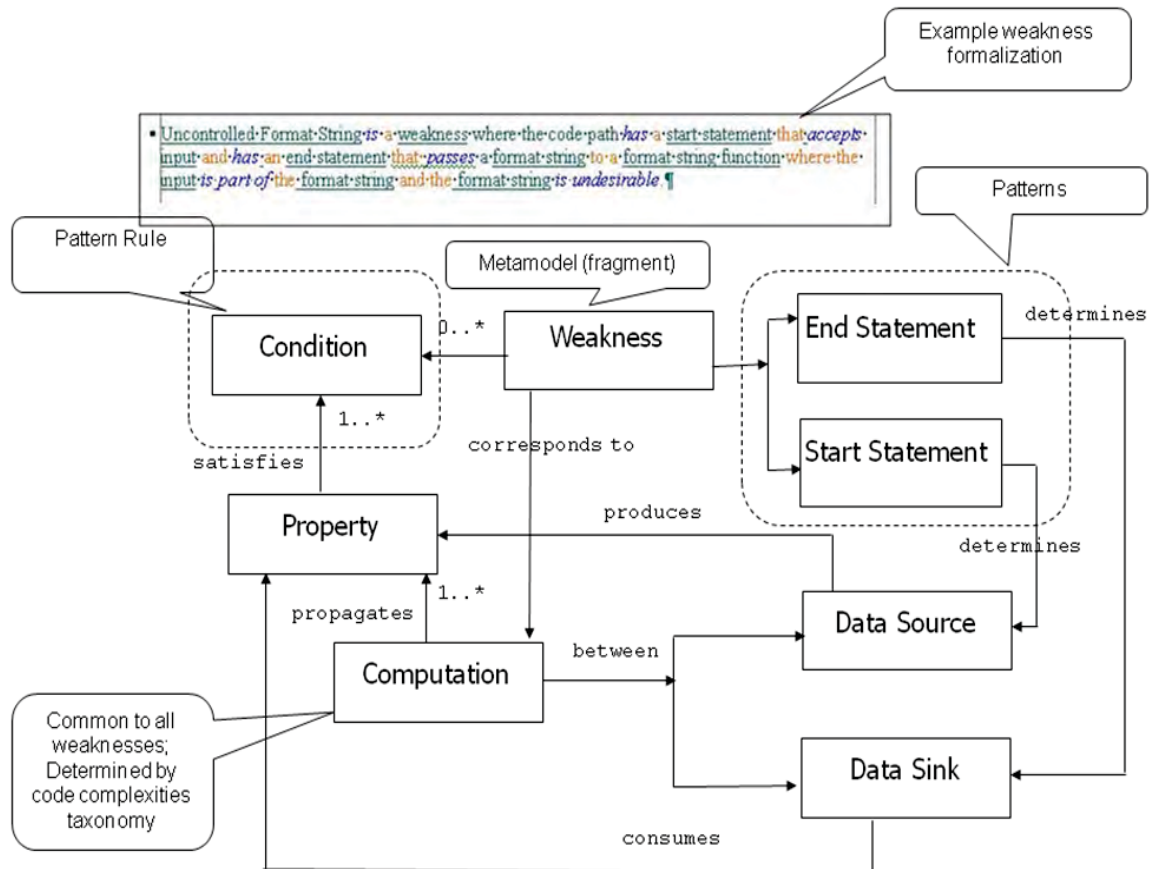


Figure 1. Weakness Conceptual Model

The logical model expands on the conceptual model to show how a pattern with a pattern rule is presented in the code and what properties are taken in account when code path is computed. The Logical Model adds concrete details to the definition of a “computation” and a “property” in white-box terms. At this point, all non discernible white box properties are removed. This is in preparation for technical formalization. The logical model is presented in Figure 2. The Weakness Logical Model shows how the SFP pattern is mapped onto both the Technical and Conceptual Formalizations.

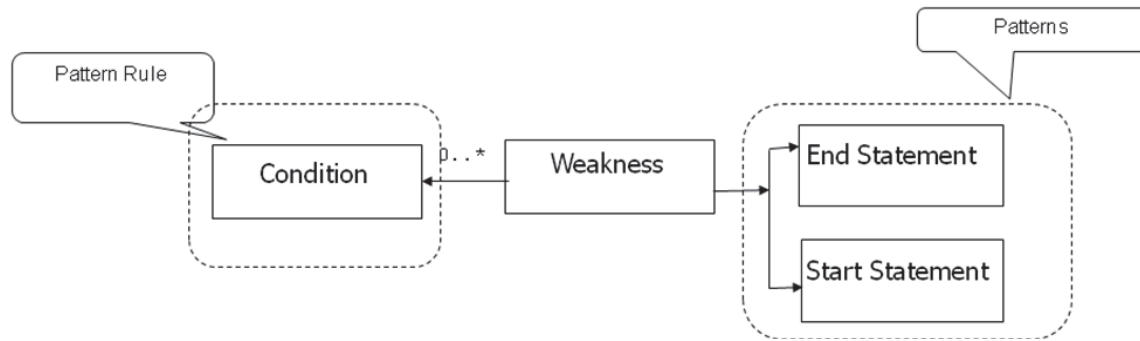


Figure 3. Weakness Definition Separated From the Corresponding Computation

The Software Fault Pattern approach focuses at the “places” in the code that are *indicators* of particular computations. In particular, the Software Fault Pattern approach focuses at the “*foot-holds*” which are places in the code that present the necessary conditions for vulnerabilities.

According to the Logical Weakness Model, each Software Fault Pattern is defined by a *pattern* which is based on some tangible foot-holds in the system’s artifacts; and some *conditions*, which involve the elements of the pattern as well as some *property* which is also based on some tangible foot-holds in the system’s artifacts.

Software Fault Patterns are arranged into common *clusters* that represent classes of computations common to large families of systems. According to the above definitions, these computations involve common safeguards, such as authentication, access control, privilege management, cryptography and data validation, as well as the common infrastructure activities, such as memory management, resource management, exception management, information management, etc.; as well as several very specific patterns where “things can go wrong”. Several computations are included into the catalog because they are common “contributors” to the real faulty computations.

Software Fault Patterns are aligned with *impact*. According to the NIST Common Vulnerability Scoring System (CVSS), impact consists of Confidentiality Impact, Integrity Impact and Availability Impact.

Confidentiality Impact “measures the impact on confidentiality of a successfully exploited vulnerability”. “Confidentiality refers to limiting information disclosure to only authorized users, as well as preventing access by, or disclosure to, unauthorized users”.

Integrity impact “measures the impact on integrity of a successfully exploited vulnerability”. “Integrity refers to the trustworthiness and guaranteed veracity of information”. Integrity impact involves modification of some system files or information.

Availability impact “measures the impact -on availability of service at the time of successfully exploited vulnerability”. “Availability refers to the accessibility of information resources. Attacks that consume bandwidth, processor cycles, or disk space all impact availability of the system.

These concepts are further elaborated in the system context and illustrated in Figure 4 below.

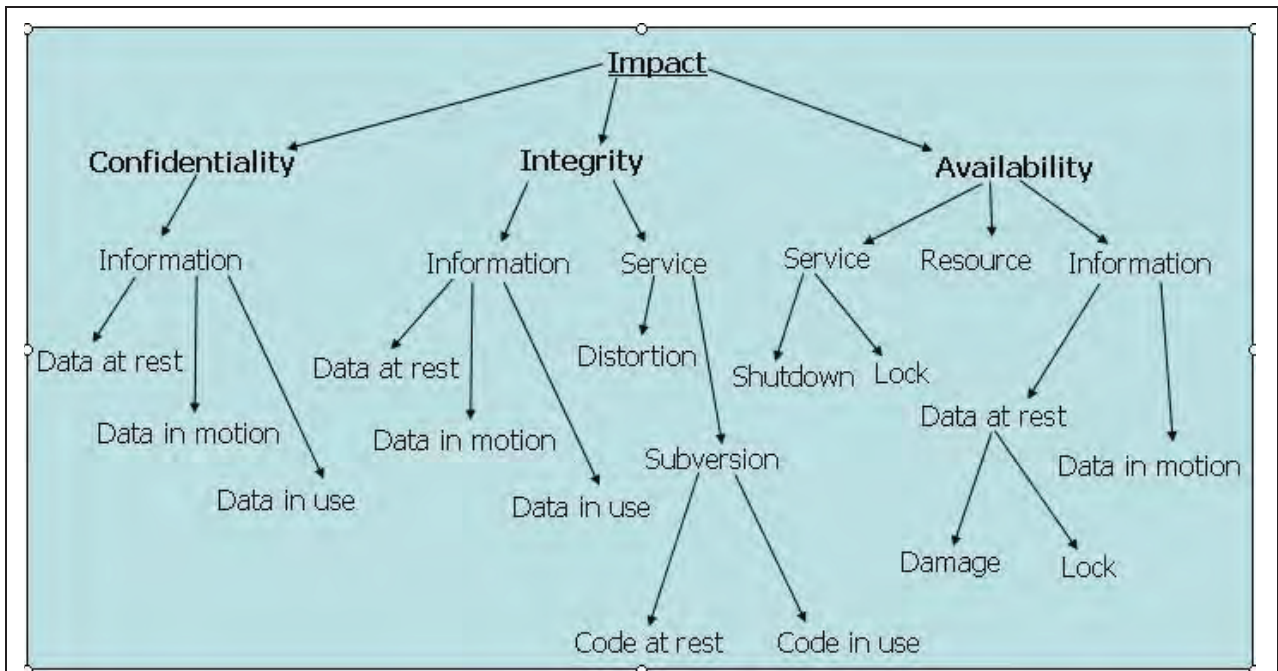


Figure 4. Classification of Impacts

1.5 Discernible and Non-Discernable Characteristics

Discernible characteristic is a property of the computation, such as the role of a data element, the role of an action or of a region, which can be expressed as a statement in the vocabulary of the system's artifacts. Discernible description of a computation is a (logical) statement that is based entirely on discernible characteristics. A non-discernible description is either ambiguous (the meaning of the definition is ill-defined, the description is not a logical statement), uses ill-defined characteristics, or uses one or more non-discernible characteristics.

The Software Fault Pattern approach uses the ISO 19506 Knowledge Discovery Metamodel (KDM) vocabulary as the baseline for the vocabulary of system's artifacts.

A non-discernable description can be turned into a discernable one by:

- Providing more clarity and precision
- Using structured English based on controlled vocabulary
- Performing additional research to better define the corresponding family of computations, and better defining the characteristics involved in the definition

- Defining additional facts that extend the currently available vocabulary of facts related to system's artifacts.

Some examples of non-discernable CWEs:

- 684 - Failure to Provide Specified Functionality - The code does not function according to its published specifications, potentially leading to incorrect usage
- 573 - Failure to Follow Specification - The software fails to follow the specifications for the implementation language, environment, framework, protocol, or platform
- 115 – Misinterpreted Input - The software misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.
- 448 - Obsolete Feature in UI - A UI function is obsolete and the product does not warn the user.

Section 2. Obtaining White Box Definitions

A naïve process for developing white-box definitions for SFPs will first identify white-box definitions for each weakness following the structured definition approach based on the Weakness Conceptual Model, and then analyze the already normalized definitions to discover common patterns and rules and determine the clusters and SFPs as the result. The disadvantage of this approach is that the input “knowledge space” is large and not normalized. A more cost-effective approach will approximate the clusters first by using automated semantic clustering tools to perform groupings of CWEs based on their informal descriptions, and then perform manual examination of the resulting clusters, and subsequently extract the SFPs. Once all the steps are completed, the white-box definitions are produced.

In other words, a cost-effective process for developing white box definitions for SFPs requires that the (informal) clusters first be identified. Once the clusters are identified, the white box descriptions are created for each SFP by following the approach based on the Weakness Conceptual Model.

On the other hand, the clusters need to be examined regardless of the CWE groupings that lead to their creation. Each cluster has to be well-defined. Each cluster should focus at a meaningful area of computations that are common to a large family of software systems. Each cluster should be determined by unique computation foot-holds and unique properties.

To achieve these objectives, we followed a two-phase process. The bottom-up phase of the process is called the **Cluster Discovery** process. The outcome of the bottom-up phase is collection of the initial families of related computations (clusters). The second phase is the **Software Fault Pattern Discovery** process based on the initial clusters. During this phase the clusters are rationalized and refactored, common patterns and their parameters are identified, and then individual software fault patterns with their parameters are described.

At the end of the top-down phase, software fault patterns are associated to the corresponding impacts.

2.1 Bottom-Up: Cluster Discovery Process

The process starts with CWEs – as de-facto space definition. CWE is a description of the “universe” of faulty computations; however the objectives of the CWE classification are somewhat different than the systematic focus at enabling automation through tangible patterns. In particular CWE has multiple places where classes overlap. Therefore one has to go through all CWEs and “discover” the actual computations – the extent of the CWE elements. In order to provide good coverage of the space, we considered all CWE weakness nodes, regardless of their parent-child relationships.

CWE descriptions were “ranked” in the following way: enough white-box content to proceed=“discernible”; not enough white-box content=“non-discernible”. Then rough

identification of the common *characteristics* of the computations from informal CWE description was performed.

Then a “distance” metric was applied to matching characteristics and initial clusters were identified. Each CWE defines a family of computations. Cluster is also a (larger) family of computations. The grouping criterion for the initial clusters is the commonality of characteristics.

This material is used to manage the set of real known patterns of “faulty computations”. Links to CWE elements are maintained in order to be able to reason about the coverage of the space.

There are 638 CWE descriptions found in CWE Release 1.4. CWE Release 1.4 contains 777 elements, but not all of them are actual weakness descriptions. Some CWE elements are designated as Views (22 elements), Categories (105 elements) and Compound Elements (12 elements). Only “weakness” elements were considered during analysis. Weakness elements are identified in the CWE 1.4 XML document as elements between tags <Weaknesses> and </Weaknesses>.

Of the 638 “weakness” elements, eight are marked as depreciated, leaving 630 for review.

In creating a SFP, the remaining actual CWE weakness descriptions are categorized by their functionality, patterns and pattern rules to create CWE Clusters. Once the clusters have been identified and refined, each CWE is examined and tagged as either:

- Suitable for formalization
- Unsuitable for formalization

For the CWEs deemed suitable for formalization, draft patterns are created. Each pattern is validated by creating semi-formal white box descriptions by adding the “start” and “end” statements as required.

After further analysis, the semi-formal descriptions are normalized, through terminology and pattern alignment between descriptions.

2.2 Top-Down: Software Fault Pattern Discovery Process

The top down process involves examination of the initial clusters to identify and describe family of computations. In other words, to discover Software Fault Patterns. CWEs are no longer involved in the top down process, although the links to the individual CWEs are maintained throughout the entire process. The objective of the top-down examination of the initial clusters is the rationalization of clusters and systematic identification of the boundaries of each family of faulty computations.

We built a full conceptual map of clusters as a step to normalized vocabulary. We examined the key characteristics of each cluster and dependencies between clusters based on these characteristics. The SFP approach is unique because it only considers

characteristics that are discernible in code artifacts, not design or system configuration artifacts. We rationalize characteristics for each cluster and further rationalize the clusters themselves.

A primary cluster is the normalized way to manage the reality of “faulty computations”. A “computation” is considered as a sequence of steps (or events). The code artifacts provide constraints to computations and therefore determine what kind of computations can occur. Computations often perform steps that are common to large families of systems. The key to the SFP approach is that there are certain “places” in the code artifacts that are identifiable indicators of particular computations. Further, in those certain places of code there exist the necessary conditions for vulnerabilities. These places are called the “foot-holds” of vulnerabilities. We further refined the primary clusters by examining the available foot-holds and created secondary clusters focused at common foot-holds. The basis of the grouping provides the common detection.

Once the outline of the catalog of software fault patterns has been finalized, we identified the individual software fault patterns. The next step is to further identify for each pattern the invariant core characteristics and the variant characteristics. The variant characteristics determine parameters. This step is called **SFP parameterization** and it is described in section 2.2.1 Parameterization of Software Fault Patterns. Once the parameterization step is performed, a generalized SFP white-box definition is developed covering invariant core and its identified parameters. Parameters are provided with sample values to illustrate their mappings to the system artifacts.

Traceability to CWEs is guaranteed through the bottom up phase. We defined the boundaries of each family of computations. Each CWE is assigned to exactly one cluster. In some situations this creates tension with the existing CWE descriptions.

At the end of this phase we aligned software fault patterns with vulnerability impacts.

2.2.1 Parameterization of Software Fault Patterns

Parameterization of SFPs is about creating a vocabulary of elementary “patterns” and discernible characteristics of these patterns, focusing at the invariants of each pattern and the variation points of each pattern. A similar process is well-known in geometry, where the language of elementary patterns includes for example, a “straight line”, a “parabola” and a “periodic line”. Using the appropriate parameters, many different lines are instances of a “straight line” and there is no need for a special term for each of them. Instead, a generic term “straight line” is used, defined by the equation $y=ax+b$, where each individual line is additionally characterized by certain values of parameters “a” and “b”. Also, the vocabulary of elementary patterns is used to define more complex patterns, or composite lines. Thus the parameterized vocabulary of “patterns” helps organize the individual observations. It should be noted, that the current CWE dictionary is at this point closer to a collection of observations than to a vocabulary of individual elementary “patterns”. The SFP approach is aimed to mitigate this deficiency.

Parameterization extends the 2-phase process described earlier. First, the individual observations that are representatives of a common family of computations (also called a cluster) are examined, and a generalized definition is provided. The generalized definition refers to the entire cluster, focusing at its invariant and the variant points. Further description of the variant points introduces additional details to the generalized definition. Variations are identified top-down in order to provide assurance of the coverage. In other words, large top-down variation groups are considered first, followed by more specific groups. Additional characteristics describing variation families are identified. Finally, parameters are mapped back to the original observations. Firstly, this allows preserving the links to the original observations (CWEs). Secondly, this allows identifying gaps in the original definitions. This process is illustrated in Figure 5 below.

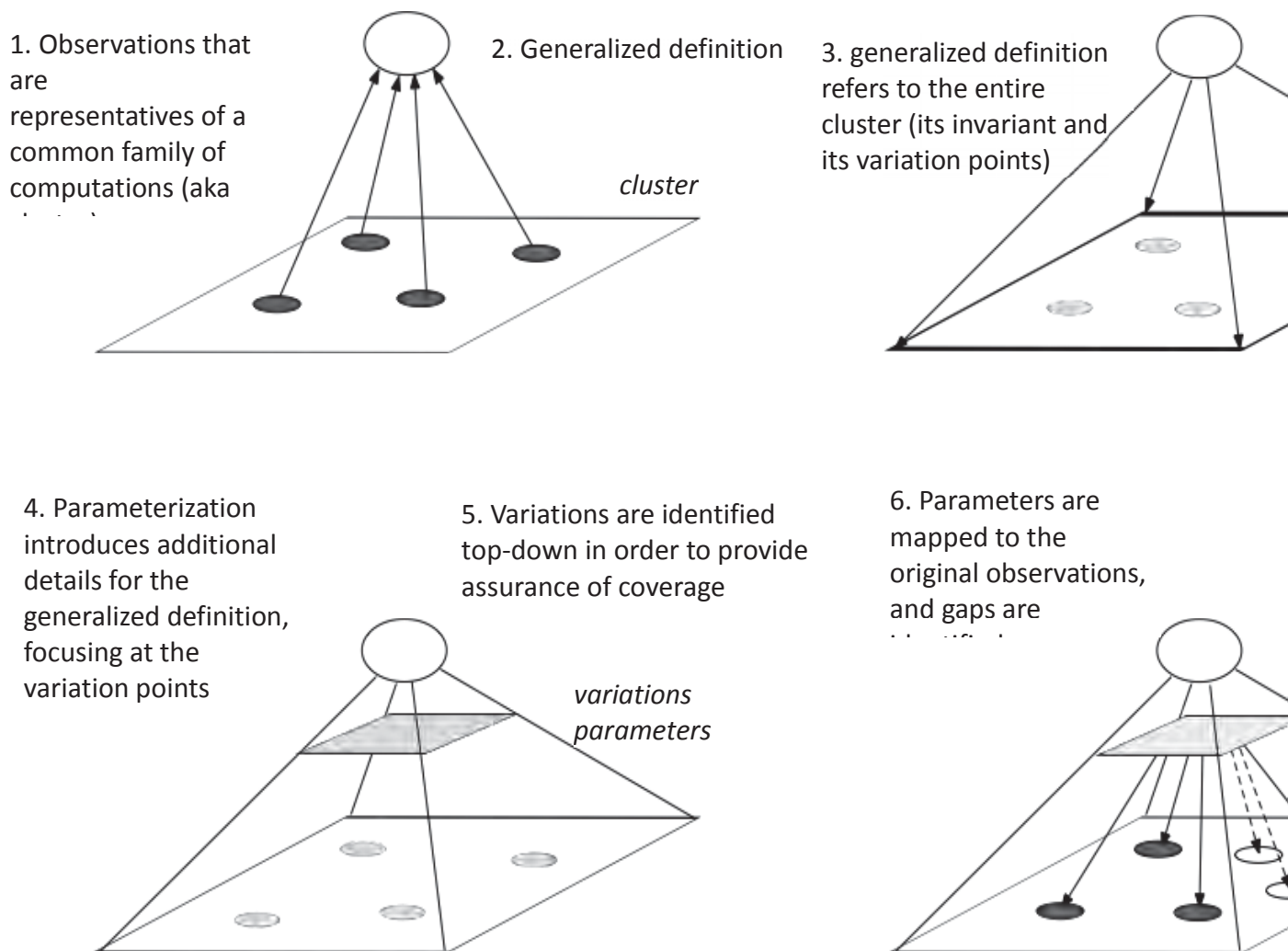


Figure 5. SFP Parameterization Process

2.3 A Quick Summary of Software Fault Pattern Extraction Process

Table 1 summarizes the tasks (in order) that were performed.

Table 1. Software Fault Pattern Extraction Process Steps

Task #	Task performed	Details
1	Each CWE weakness description has been analyzed.	Analyzed CWEs have been entered into the spreadsheet accompanying this document. CWE version 1.5 has been analyzed. There are total 638 CWE weaknesses, including 8 marked as deprecated.
2	Each CWE weakness description has been assigned a rank [1..5] indicating the quality of the white-box content in the original weakness description	Ranks have been entered into the spreadsheet accompanying this document. <ul style="list-style-type: none">• Rank 5 means “The content of this CWE weakness description is based directly on the well-understood discernible white-box properties”.• Rank 4 means “The content of this CWE weakness description is based on discernible white-box properties”.• Rank 3 means “The content of this CWE weakness description is based on discernible white-box properties or properties that are believed to be derivable from them”.• Rank 2 means “The content of this CWE weakness description involves properties that are not derivable from discernible white-box properties”.• Rank 1 means “The content of this CWE weakness description is not discernible”
3	Meaningful groupings of CWE weaknesses have been suggested	21 primary clusters have been suggested. Primary clusters are based on the common functionality of the corresponding computation and are to a large extent correlated with the use of the common key terminology, orthogonal between the clusters. Primary clusters are further subdivided into 80 secondary clusters. Cluster names have been entered into the

		<p>spreadsheet accompanying this document.</p> <p>The document contains a short description of each cluster</p>
4	Each cluster has been given a short description	Description of each primary cluster includes the common characteristics of the cluster, associated clusters, and descriptions of each secondary cluster.
5	Each CWE has been associated with exactly one primary and secondary cluster	<p>Each secondary cluster contains the list of individual CWEs in this cluster.</p> <p>Cluster names have been entered for each CWE into the spreadsheet accompanying this document.</p>
6	Software Fault Patterns have been identified	36 Software Fault Patterns have been identified.
7	Each CWE with rank 5,4 or 3 (discernible CWEs) has been associated with exactly one Software Fault Pattern	Only CWEs ranked as 3, 4 and 5 contribute and maintain the link to Software Fault Patterns
8	Each Software Fault Pattern has been given a white-box content description	Software Fault Pattern follows the Weakness Logical Model and identifies the end and start statement, the condition and the property.
9	Software Fault Patterns have been parameterized	This step is to further identify for each pattern the invariant core characteristics and the variant characteristics. The variant characteristics determine parameters.
10	CWEs have been represented as specializations of the corresponding Software Fault Pattern	Software Fault Patterns identify certain elements as parameters. Individual CWEs can be represented as specializations of a common software fault pattern with specific values of Software Fault Pattern parameters.

Section 3. Summary of Clusters and Software Fault Patterns

The rest of this document focuses on further analysis and revision of identified clusters and their software fault patterns.

Summary:

The following table (Table 2) provides the summary of the Clusters and Software Fault Patterns.

- Column 1 describes the Primary Cluster.
- Column 2 enumerates all secondary clusters of the given primary cluster (one per row).
- Column 3 (# of CWEs) shows the number of CWEs in the secondary cluster.
- Column 4 (Primary CWE Totals) shows the total number of the CWEs for the entire primary cluster.
- Column 5 (Pattern & Condition Available?) provides a brief description of the secondary cluster: “yes” means that all CWEs in the secondary cluster are discernible; “singular” means that the cluster contains a single discernible CWE; “partial” means that there are some discernible and some non-discernible CWEs in the cluster; “no” means that the cluster contains only non-discernible CWEs.
- Column 6 (WB CWE) shows the number of discernible CWEs in the cluster (CWEs with white-box definitions)
- Column 7 (SFP #) provides the reference to the corresponding software fault pattern.

Table 2. Software Fault Clusters Summary Table

<u>Software Fault Clusters</u>						
<u>Primary</u>	<u>Secondary</u>	<u># of CWEs</u>	<u>Primary CWE Totals</u>	<u>Pattern & Condition Available?</u>	<u>White- Box CWE</u>	<u>SFP #</u>
Risky Values			31			
	Glitch in computation	31		partial	27	SFP1
Unused entities			3			
	Unused entities	3		yes	3	SFP2
API			28			
	Use of an improper API	28		partial	20	SFP3
Exception Management			27			
	Unchecked status condition	17		partial	13	SFP4
	Ambiguous exception type	2		yes	2	SFP5
	Incorrect exception behavior	8		partial	3	SFP6
Memory Access			20			
	Faulty pointer use	3		yes	3	SFP7
	Faulty buffer access	11		yes	11	SFP8
	Faulty string expansion	2		yes	2	SFP9
	Incorrect buffer length computation	3		partial	2	SFP10
	Improper NULL termination	1		singular	1	SFP11
Memory Management			5			
	Faulty memory release	5		yes	5	SFP12
Resource Management			17			
	Unrestricted consumption	4		partial	3	SFP13
	Failure to release resource	7		yes	7	SFP14
	Faulty resource use	2		yes	2	SFP15
	Life cycle	4		no	0	-
Path Resolution			51			
	Path traversal	43		partial	38	SFP16
	Failed chroot jail	1		singular	1	SFP17
	Link in resource name resolution	7		partial	4	SFP18
Synchronization			22			

<u>Software Fault Clusters</u>						
<u>Primary</u>	<u>Secondary</u>	<u># of CWEs</u>	<u>Primary CWE Totals</u>	<u>Pattern & Condition Available?</u>	<u>White- Box CWE</u>	<u>SFP #</u>
	Missing lock	13		partial	10	SFP19
	Race condition window	5		partial	4	SFP20
	Multiple locks/unlocks	3		yes	3	SFP21
	Unrestricted lock	1		singular	1	SFP22
Information Leak			96			
	Exposed data	76		partial	38	SFP23
	State disclosure	7		no	0	-
	Exposure through temporary file	3		no	0	-
	Other exposures	7		no	0	-
	Insecure session management	3		no	0	-
Tainted Input			138			
	Tainted input to command	87		partial	68	SFP24
	Tainted input to variable	8		yes	8	SFP25
	Composite tainted input	0		no	0	SFP26
	Faulty input transformation	15		no	0	-
	Incorrect input handling	17		no	0	-
	Tainted input to environment	11		partial	3	SFP27
Entry Points			11			
	Unexpected access points	11		yes	11	SFP28
Authentication			43			
	Authentication bypass	10		no	0	-
	Faulty endpoint authentication	11		partial	6	SFP29
	Missing endpoint authentication	2		yes	2	SFP30
	Digital certificate	6		no	0	-
	Missing authentication	2		yes	2	SFP31
	Insecure authentication policy	6		no	0	-
	Multiple binds to the same port	1		singular	1	SFP32
	Hardcoded sensitive data	4		partial	2	SFP33
	Unrestricted authentication	1		singular	1	SFP34
Access Control			16			

<u>Software Fault Clusters</u>						
<u>Primary</u>	<u>Secondary</u>	<u># of CWEs</u>	<u>Primary CWE Totals</u>	<u>Pattern & Condition Available?</u>	<u>White- Box CWE</u>	<u>SFP #</u>
	Insecure resource access	4		partial	2	SFP35
	Insecure resource permissions	7		no	0	-
	Access management	5		no	0	-
Privilege			12			
	Privilege	12		partial	1	SFP36
Channel			13			
	Channel Attack	8		no	0	-
	Protocol error	5		no	0	-
Cryptography			13			
	Broken cryptography	5		no	0	-
	Weak cryptography	8		no	0	-
Malware			11			
	Malicious code	8		no	0	-
	Covert channel	3		no	0	-
Predictability			15			
	Predictability	15		no	0	-
UI			14			
	Feature	7		no	0	-
	Information loss	4		no	0	-
	Security	3		no	0	-
Other			46			
	Architecture	11		no	0	-
	Design	29		no	0	-
	Implementation	5		no	0	-
	Compiler	1		no	0	-
			632		310	36

From the above table, it can be seen that 632 CWEs have been categorized. Since the total deprecated CWEs are 8, then the total CWEs defined as weaknesses total 640.

In addition, there are:

- 21 Primary Clusters
- 62 Secondary Clusters
- 310 discernible CWEs
- 36 unique Software Fault Patterns identified.

Section 4. Clusters and Software Fault Patterns

A “cluster” is a family of computations that share certain common characteristics. Identifying vulnerability clusters is the first step towards identifying software fault patterns. In particular, all software fault patterns are located within a certain cluster. A two-level hierarchical organization of clusters is suggested. First there are 21 primary clusters. Primary clusters are identified based on the common functionality of CWEs. Second, each primary cluster is further subdivided into one or more secondary clusters, based on the common patterns. As the result, each CWE is associated with exactly one primary and exactly one secondary cluster. This association does not depend on the discernibility of the CWE.

Discernible CWEs in a secondary cluster contribute to a software fault pattern.

The following sub-sections describe 21 primary and 62 secondary clusters extracted by examining 632 CWEs and their associated Software Fault Patterns.

4.1 Primary Cluster: Risky Values

This cluster of weaknesses relates to the basic uses of numeric values in software systems. The characteristics of “Risky Values” involve:

- Creation of numeric values
- Operations involving numeric values
- Exceptions raised by numeric operations

Certain values are unexpected in certain contexts, while they are perfectly legitimate in most other contexts. These are situations like the use of an uninitialized value or division by zero. These usages may have impact and raise exceptions on some platforms. Also faulty (unexpected) numeric values may be generated as the result of certain numeric operations, like type cast or arithmetic. These situations do not have explicit impact. However, faulty values may flow into the following contexts:

- Buffer access operations
- Simple conditions
- Loop conditions

Through these characteristics, the Risky Values cluster is associated with the following clusters:

- Memory Access
- Exception Management

- API
- Clusters that involve conditions
 - Authentication
 - Access Control
 - Privilege
 - Synchronization
 - Resource Management

This cluster contains 31 CWEs. 27 of these CWEs are all based on discernible properties and are therefore covered by a software fault pattern.

The Risky Values cluster has a single secondary cluster „Glitch in computation“.

4.1.1 Secondary Cluster: Glitch in Computation

This cluster describes a large family of computations that violate naïve assumptions about the resulting data. Each computation involves an identifiable operation that processes the input data of certain datatypes (integer, Boolean, etc.) and produces a certain resulting value. Under certain conditions involving the input data, the resulting value takes an anomalous value, which can violate naïve assumptions. The particular glitch (or the anomalous resulting value) may be described, for example as an overflow, an underflow, loss of significant digits, exception, etc. One can say, that certain input data, satisfying the glitch conditions, is inappropriate for the given operation. The impact of the glitch in computation is the potential loss of integrity of the data in use or the loss of the availability of the corresponding service.

However, a more common situation is that the anomalous data is propagated into another more security sensitive context and causes more serious injury. For example, the incorrect data can be used downstream as the length of a buffer or as the length of data in a buffer allocation operation or in a buffer access operation, or it can be used as a throttle value in resource allocation loop, etc. The anomalous resulting value may also cause unchecked exception.

The “Glitch in computation” cluster is further subdivided into the following 9 groups for easier understanding and management:

- a. Wrap around error – this group covers unconstrained numeric operations where the data value can be proven to exceed the boundary value of the corresponding datatype.
- b. Unsafe type conversion – this group covers type conversion operations where the data value can be proven to change unexpectedly

- c. Incorrect pointer scaling – this group contains a very specific operation involving a combination of pointer arithmetic a type conversion, which is known to correlate with field issues.
- d. Use of an Uninitialized value – this group covers a unique scenario where an “uninitialized” data value is used
- e. Divide by zero – this group covers a unique scenario where a value 0 is used as a divisor of the divide operation
- f. Suspicious condition – this group covers suspicious conditions that always select a single branch, or may involve incorrect assignments or may not account for short circuit condition logic
- g. Incorrect operations on non-serializable objects – this group covers several related faulty computations involving non-serialized objects
- h. Incorrect parameters to an API – this group covers common weaknesses related to parameter passing to and from an API call
- i. Faulty pointer creation – this group is closely related to faulty pointer use, but focuses at the scenarios where faulty pointers are usually created (rather than the places where they are used)

4.1.1.1 SFP1 Glitch in Computation

Software Fault Pattern – a weakness where the code path has all of the following:

- End statement that performs an identifiable operation on data producing some actual value of a datatype and
- The data is inappropriate to the operation resulting in the value that is unexpected for the datatype and the operation

The parameters of this family of computation include the variation of injury, the operation, and the particular conditions of the input data that results in anomalous resulting value. For the sample values assigned to these parameters and CWE mappings refer to spreadsheet accompanying this document.

The following sections describe each group of computations and the corresponding CWEs in more detail.

4.1.1.2 Wrap around Error

This group covers unconstrained numeric operations where the data value can be proven to exceed the boundary value of the corresponding datatype.

This group has 3 CWEs. All CWEs are discernible.

The following table lists all discernible CWEs that contribute to this software fault pattern:

128	Wrap-around Error	Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.
190	Integer Overflow or Wraparound	The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.
191	Integer Underflow (Wrap or Wraparound)	The product subtracts one value from another, such that the result is less than the minimum allowable integer value, which produces a value that is not equal to the correct result.

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.1.1.3 Unsafe Type Conversion

This group covers type conversion operations where the data value can be proven to change in often unexpected ways.

This group has 6 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs that contribute to this group:

194	Unexpected Sign Extension	The software performs an operation on a number that causes it to be sign extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.
195	Signed to Unsigned Conversion Error	A signed-to-unsigned conversion error takes place when a signed primitive is used as an unsigned value, usually as a size variable.
196	Unsigned to Signed Conversion Error	An unsigned-to-signed conversion error takes place when a large unsigned primitive is used as a signed value.

197	Numeric Truncation Error	Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.
681	Incorrect Conversion between Numeric Types	When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.
704	Incorrect Type Conversion or Cast	The software does not correctly convert an object, resource or structure from one type to a different type.

4.1.1.4 Incorrect Pointer Scaling

This group contains a very specific operation involving a combination of pointer arithmetic a type conversion, which is known to generate unexpected values and correlates with field issues.

This group has 1 CWE. The CWE in the group is discernible.

The following table lists all discernible CWEs that contribute to this group:

468	Incorrect Pointer Scaling	In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.
------------	---------------------------	--

4.1.1.5 Use of an Uninitialized Value

This group covers the use of the value of an uninitialized value. An Uninitialized value only manifests itself when it is used in an operation (other than reassignment of passing as a by value parameter to other function).

This group has 2 CWEs. All CWEs are discernible.

The following table lists all discernible CWEs that contribute to this group:

456	Missing Initialization	The software does not initialize critical variables, which causes the execution environment to use unexpected values.
------------	------------------------	---

457	Use of Uninitialized Variable	The code uses a variable that has not been initialized, leading to unpredictable or unintended results.
------------	-------------------------------	---

4.1.1.6 Divide by Zero

This group covers the use of a value zero as the second operand of the division operation.

This group has 1 discernible CWE.

The following table lists all discernible CWEs that contribute to this group:

369	Divide By Zero	The product divides a value by zero.
------------	----------------	--------------------------------------

4.1.1.7 Suspicious Condition

This group covers suspicious conditions that always select a single branch, or may involve incorrect assignments or may not account for a short circuit in condition logic.

This group has 6 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs that contribute to this group:

481	Assigning instead of Comparing	The code uses an operator for assignment when the intention was to perform a comparison.
486	Comparison of Classes by Name	The program compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.
570	Expression is Always False	The software contains an expression that will always evaluate to false.
571	Expression is Always True	The software contains an expression that will always evaluate to true.
597	Use of Wrong Operator in String Comparison	The product uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead.
768	Incorrect Short Circuit Evaluation	The software contains conditionals with multiple logical expressions where one or more of the non-leading logical expressions produce side effects that may not be executed due to short circuiting logic. This may lead to an unexpected state in the program after the execution of the conditional.

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.1.1.8 Incorrect Operation on Non-serializable Object

This group covers several related misuse scenarios for non-serialized objects.

This group has 2 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs that contribute to this group:

579	J2EE Bad Practices: Non-serializable Object Stored in Session	The application stores a non-serializable object as an HttpSession attribute, which can hurt reliability.
594	J2EE Framework: Saving Unserializable Objects to Disk	When the J2EE container attempts to write unserializable objects to disk there is no guarantee that the process will complete successfully.

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.1.1.9 Incorrect Parameters to an API

This group covers common weaknesses related to parameter passing to and from an API call.

This group has 7 CWEs. 3 CWEs in the group are discernible. 4 CWEs are non-discernible.

The following table lists all discernible CWEs that contribute to this group:

475	Undefined Behavior for Input to API	The behavior of this function is undefined unless its control parameter is set to a specific value.
685	Function Call With Incorrect Number of Arguments	The software calls a function, procedure, or routine, but the caller specifies too many arguments, or too few arguments, leading to undefined behavior and resultant weaknesses.
686	Function Call With Incorrect Argument Type	The software calls a function, procedure, or routine, but the caller specifies an argument that is the wrong data type, leading to resultant weaknesses.

Non-discernible CWEs in this cluster:

628	Function Call with Incorrectly Specified Arguments	The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.
683	Function Call With Incorrect Order of Arguments	The software calls a function, procedure, or routine, but the caller specifies the arguments in an incorrect order, leading to resultant weaknesses.
687	Function Call With Incorrectly Specified Argument Value	The software calls a function, procedure, or routine, but the caller specifies an argument that contains the wrong value, leading to resultant weaknesses.
688	Function Call With Incorrect Variable or Reference as Argument	The software calls a function, procedure, or routine, but the caller specifies the wrong variable or reference as one of the arguments, leading to undefined behavior and resultant weaknesses.

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.1.1.10 Faulty Pointer Creation

This group is closely related to faulty pointer use, but focuses at the scenarios where faulty pointers are usually created (rather than the places where they are used).

This group has 4 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs that contribute to this group:

466	Return of Pointer Value Outside of Expected Range	A function can return a pointer to memory that is outside of the buffer that the pointer is expected to reference.
562	Return of Stack Variable Address	A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash.
587	Assignment of a Fixed Address to a Pointer	The software sets a pointer to a specific address other than NULL or 0.

NOTE: there are related software fault patterns:

SFP7 Faulty pointer use (uses of incorrect data items)

4.2 Primary Cluster: Unused Entities

This cluster covers a general case of unused entities in code, including unused procedures or variables.

This cluster contains 3 CWEs. These CWEs are based on discernible properties and are therefore covered by software fault patterns.

The Unused Entities cluster includes a single secondary cluster:

- Unused entities – this cluster covers a general case of unused code that is defined for entire entities, like unused procedures or variables

4.2.1 Secondary Cluster: Unused Entities

This cluster covers a general case of unused entities in code, including unused procedures or variables. “Usage” of an entity (be it a variable or a procedure) is defined in terms of *any* kinds of incoming relationships. Note that these relationships are different for variables and procedures. Only “first order” uses are considered, for example, if procedure “A” is being called only from another procedure “B”, which is in itself “unused”, procedure “A” is not considered “unused”. (NOTE: This particular cluster has direct applicability to the aviation industry where “Dead Code” is not allowed in avionics safety critical software.)

This cluster has 3 CWEs. All CWEs in the cluster are discernible.

4.2.1.1 SFP2 Unused Entities

Software Fault Pattern – an **entity** that does not have incoming usage relationships

NOTE: This software fault pattern does not involve a code path.

The parameter of this software fault pattern is: entity. The software fault pattern is parameterized with the particular kind of entity to report.

The following table lists all discernible CWEs that contribute to this software fault pattern:

482	Comparing instead of Assigning	The code uses an operator for comparison when the intention was to perform an assignment.
561	Dead Code	The software contains dead code, which can never be executed.
563	Unused Variable	The variable's value is assigned but never used, making it a dead store.

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.3 Primary Cluster: API

This cluster of weaknesses relates to the use of Application Programming Interfaces (API) which are calls to encapsulated functionality provided by the runtime platform. Common characteristics of the weaknesses in this cluster involve:

- Known incompatibilities between different parts of the runtime platform
- API, commonly known as dangerous or otherwise problematic
- Common problems related to parameter passing to API calls

The common pattern for this group of weaknesses involves a call to an external procedure or method which may also involve some parameter passing.

This cluster contains 28 CWEs. 20 of these CWEs are based on discernible properties and are therefore covered by software fault patterns.

The API cluster includes a single secondary cluster:

- Use of an improper API – this cluster covers common patterns involving only the identity of the API being called.

4.3.1 Secondary Cluster: Use of an Improper API

This cluster covers common patterns involving only the identity of the API being called.

This cluster has 28 CWEs. 20 CWEs in the cluster are discernible. 8 CWEs are non-discernible.

4.3.1.1 SFP3 Use of an Improper API

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs an API call where the call is not appropriate for the given platform

NOTE: This software fault pattern considers only the name of the API. It considers the knowledge base to determine the appropriateness of the API call on the given platform. Some APIs are annotated in the knowledge base, while other cases are described as incompatibility between two platform containers.

The following table lists all discernible CWEs that contribute to this software fault pattern:

111	Direct Use of Unsafe JNI	When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java.
242	Use of Inherently Dangerous Function	The program calls a function that can never be guaranteed to work safely.
245	J2EE Bad Practices: Direct Management of Connections	The J2EE application directly manages connections, instead of using the container's connection management facilities.
246	J2EE Bad Practices: Direct Use of Sockets	The J2EE application directly uses sockets instead of using framework method calls.
382	J2EE Bad Practices: Use of System.exit()	A J2EE application uses System.exit(), which also shuts down its container.
383	J2EE Bad Practices: Direct Use of Threads	Thread management in a Web application is forbidden in some circumstances and is always highly error prone.
474	Use of Function with Inconsistent Implementations	The code uses a function that has inconsistent implementations across operating systems and versions, which might cause security-relevant portability problems.
477	Use of Obsolete Functions	The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.
479	Unsafe Function Call from a Signal Handler	The program has a signal handler that calls an unsafe function, leading to unpredictable results.
558	Use of getlogin() in Multithreaded Application	The application uses the getlogin() function in a multithreaded context, potentially causing it to return incorrect values.
574	EJB Bad Practices: Use of Synchronization Primitives	The program violates the Enterprise JavaBeans (EJB) specification by using thread synchronization primitives.
575	EJB Bad Practices: Use of AWT Swing	The program violates the Enterprise JavaBeans (EJB) specification by using AWT/Swing.
576	EJB Bad Practices: Use of Java I/O	The program violates the Enterprise JavaBeans (EJB) specification by using the java.io package.

577	EJB Bad Practices: Use of Sockets	The program violates the Enterprise JavaBeans (EJB) specification by using sockets.
578	EJB Bad Practices: Use of Class Loader	The program violates the Enterprise JavaBeans (EJB) specification by using the class loader.
589	Call to Non-ubiquitous API	The software uses an API function that does not exist on all versions of the target platform. This could cause portability problems or inconsistencies that allow denial of service or other consequences.
676	Use of Potentially Dangerous Function	The program invokes a potentially dangerous function that could introduce vulnerability if it is used incorrectly, but the function can also be used safely.
617	Reachable Assertion	The product contains an assert() or similar statement that can be triggered by an attacker, which leads to an application exit or other behavior that is more severe than necessary.
572	Call to Thread run() instead of start()	The program calls a thread's run() method instead of calling start(), which causes the code to run in the thread of the caller instead of the callee.
586	Explicit Call to Finalize()	The software makes an explicit call to the finalize() method from outside the finalizer.

Non-discernible CWEs in this cluster:

227	Failure to Fulfill API Contract ('API Abuse')	The software uses an API in a manner contrary to its intended use.
432	Dangerous Handler not Disabled During Sensitive Operations	The application does not properly clear or disable dangerous handlers during sensitive operations.
439	Behavioral Change in New Version or Environment	A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.
440	Expected Behavior Violation	A feature, API, or function being used by a product behaves differently than the product expects.
573	Failure to Follow Specification	The software fails to follow the specifications for the implementation language, environment, framework, protocol, or platform.

684	Failure to Provide Specified Functionality	The code does not function according to its published specifications, potentially leading to incorrect usage.
695	Use of Low-Level Functionality	The software uses low-level functionality that is explicitly prohibited by the framework or specification under which the software is supposed to operate.
758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

4.4 Primary Cluster: Exception Management

This cluster of weaknesses relates to the basic management of exceptions and other status conditions usually arising from the interactions with the runtime platform. Common characteristics of the weakness in this cluster include:

- Actual state of computation
- Expected state of the computation
- Status condition
- Status condition check
- Operation that report status
- Code region coordinated with state.

This cluster is fundamental to the various computations performed by software systems. Therefore this cluster is associated with virtually all other clusters either through the use of APIs and primitive operations (operations that report status) or through condition checks.

This cluster contains 27 CWEs. 18 CWEs in this cluster are based on discernible properties and are therefore covered by software fault patterns, while others involve non-discernible properties in their descriptions and do not contribute to any software fault patterns until more white-box scenarios for these weaknesses are discovered and agreed upon by the community.

The Exception Management cluster includes the following 3 secondary clusters:

Unchecked status condition – this cluster covers common situations where the identity of the status condition is lost resulting resource operations that may be

- performed at the incorrect objects or entire code regions where the actual state of some object is not coordinated with the expected state of this object.
- j. Ambiguous exception type – this cluster covers several common patterns where the so-called exception signature includes more general exceptions than the ones actually generated by the corresponding code region.
- k. Incorrect exception behavior – this cluster covers several common situations where the exception handling behavior is problematic however this cluster almost entirely lacks sufficient white-box content

4.4.1 Secondary Cluster: Unchecked Status Condition

This cluster covers common situations where the identity of the status condition is lost resulting resource operations that may be performed at the incorrect objects or entire code regions where the actual state of some object is not coordinated with the expected state of this object.

This cluster has 17 CWEs. 13 CWEs in the cluster are discernible. 4 CWEs are non-discernible.

4.4.1.1 SFP4 Unchecked Status Condition

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that produces status condition
- an end statement incorrectly acts on the status condition
- where “incorrect act” is defined as exactly one of the following:
 - status condition never obtained and used
 - status condition obtained but not used
 - status condition incorrectly validated such as that actual and expected status mismatch

The following table lists all discernible CWEs that contribute to this software fault pattern:

248	Uncaught Exception	Failing to catch an exception thrown from a dangerous function can potentially cause the program to crash.
252	Unchecked Return Value	The software does not check the return value from a method or function, which can prevent it from

		detecting unexpected states and conditions.
253	Incorrect Check of Function Return Value	The software incorrectly checks a return value from a function, which prevents the software from detecting errors or exceptional conditions.
273	Improper Check for Dropped Privileges	The software attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded.
280	Improper Handling of Insufficient Permissions or Privileges	The application does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the application in an invalid state.
390	Detection of Error Condition Without Action	The software detects a specific error, but takes no actions to handle the error.
391	Unchecked Error Condition	Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.
394	Unexpected Status Code or Return Value	The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.
431	Missing Handler	A handler is not available or implemented.
600	Failure to Catch All Exceptions in Servlet	A Servlet fails to catch all exceptions, which may reveal sensitive debugging information.
665	Improper Initialization	The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.
478	Missing Default Case in Switch Statement	The code does not have a default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.
484	Omitted Break Statement in Switch	The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.

Non-discernible CWEs in this cluster:

372	Incomplete Internal State Distinction	The software does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.
395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.
754	Improper check for Exceptional Conditions	The software fails to check or improperly checks for an exceptional condition.
755	Improper Handling of Exceptional Conditions	The software fails to handle or improperly handles an exceptional condition.

Comments:

CWE 252 status condition lost all references or status condition was not checked.

CWE 253 there is a mismatch between the required precondition and the status condition.

CWE 273 where the status condition is related to dropped privileges and the status. The condition lost all references or is not checked or there is a mismatch between the required precondition and the status condition.

CWE 394 where the status condition lost all references or is not checked or there is a mismatch between the required precondition and the status condition.

Note: 394 is a union of 252 and 253.

CWE 248, 431, 600 where the status condition is reported as exception and is lost at the entry point of the application (no appropriate catch in the entire method call stack).

This software fault pattern uses the following knowledge base:

- <platform> <resource> <api> <status condition> how status condition is returned (return value, argument, global variable, exception)
- <Platform> <resource> <api> <status condition> <condition> <state> the post condition; what are the value ranges of the status condition, and symbolic name of the corresponding state
- Meaning that on the given platform, a certain api call returns status condition and for the given value of condition the resource is in a given state
- <platform> <resource> <api> <precondition> <state> the precondition; symbolic name of the acceptable state.

NOTE: This knowledge base also handles resource initialization and several other issues.

4.4.2 Secondary Cluster: Ambiguous Exception Type

This cluster covers several common patterns where the so-called exception signature includes more general exceptions than the ones actually generated by the corresponding code region.

This cluster has 2 CWEs. All CWEs in the cluster are discernible.

4.4.2.1 SFP5 Ambiguous Exception Type

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that requires exception signature where the exception signature is more general than the corresponding exception profile

Where:

- Exception profile is the set of exceptions thrown by a code fragment
 $EP = \{e_1, \dots, e_k\}$
- Exception signature is the set of exceptions declared for the try-block (in which case it should match the exception profile of the try-block) or at the method declaration (in which case it should match the exception profile of the entire method) $ES = \{s_1, \dots, s_l\}$
- Exception signature (ES) is more general than the exception profile (EP) of the corresponding code fragment if ES contains s which is a supertype of one or more e_i in EP.

The following table lists all discernible CWEs that contribute to this software fault pattern:

396	Declaration of Catch for Generic Exception	Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.
397	Declaration of Throws for Generic Exception	Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Comments:

CWE 396 exception signature at the try-block

CWE 397 exception signature at the method declaration

4.4.3 Secondary Cluster: Incorrect exception Behavior

This cluster covers several common situations where the exception handling behavior is problematic however this cluster almost entirely lacks sufficient white-box content.

This cluster has 8 CWEs. 3 CWE in the cluster are discernible. There are 5 non-discernible CWEs in this cluster.

4.4.3.1 SFP6 Incorrect Exception Behavior

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that assigns incorrect value to status condition
- an end statement that uses incorrect value of status condition

The following table lists all discernible CWEs that contribute to this software fault pattern:

392	Failure to Report Error in Status Code	The software encounters an error but does not return a status code or return value to indicate that an error has occurred.
393	Return of Wrong Status Code	A function or operation returns an incorrect return value or status code that does not indicate an error, but causes the product to modify its behavior based on the incorrect result.
584	Return Inside Finally Block	The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded.

Non-discernible CWEs in the incorrect exception behavior cluster:

455	Non-exit on Failed Initialization	The software does not exit or otherwise modify its operation when security-relevant errors occur during initialization, such as when a configuration file has a format error, which can cause the software to execute in a less secure fashion than intended by the administrator.
460	Improper Cleanup on Thrown Exception	The product does not clean up its state or incorrectly cleans up its state when an exception is thrown, leading to unexpected state or control flow.

544	Failure to Use a Standardized Error Handling Mechanism	The software does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.
636	Not Failing Securely ('Failing Open')	When the product encounters an error condition or failure, its design requires it to fall back to a state that is less secure than other options that are available, such as selecting the weakest encryption algorithm or using the most permissive access control restrictions.
703	Failure to Handle Exceptional Conditions	The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.

4.5 Primary Cluster: Memory Access

This cluster of weaknesses relates to access to memory buffers. Common characteristics of this cluster include:

- Buffer, including stack and heap buffers; static and dynamic buffers
- Buffer identity (pointer, name)
- Buffer access operations, including implicit buffer access (also known as string expansion)
- Operations involving buffer
- Pointer uses, including pointer export

Through these characteristics this cluster is associated with the following clusters:

- Memory management (through buffer)
- Information leak (through buffer cleanup)
- Tainted input and risky values (through properties of buffer access operation, such as data length, index; through buffer properties such as buffer length)
- Synchronization (through buffers that can be shared resources)
- API (through passing faulty pointers as parameters)

- Exception management (through buffer status and other status conditions involved in various buffer operations).

This cluster contains 20 CWEs. 19 of the CWEs in this cluster are based on discernible properties and are therefore covered by few software fault patterns. There is 1 non-discernible CWE in this cluster.

The “Memory access” cluster includes the following 5 secondary clusters:

Faulty pointer use – this cluster covers common scenarios of using incorrect pointers to buffers

- l. Faulty buffer access – this cluster covers the common scenarios related to various buffer overflows, underflows and related weaknesses
- m. Faulty string expansion – this cluster covers scenarios related to the use of certain API calls that involve implicit buffers and may lead to buffer overflows
- n. Incorrect buffer length computation – this cluster covers scenarios related to several known situations where the length of a buffer is incorrectly computed
- o. Improper NULL termination – this cluster covers scenarios related to several operations involving buffer which may lead to buffer overflows due to mismatch in data terminators within the data stored in the buffer

4.5.1 Secondary Cluster: Faulty Pointer Use

This cluster covers common scenarios of using incorrect pointers to buffers.

This cluster has 3 CWEs. All CWEs in the cluster are discernible.

4.5.1.1 SFP7 Faulty Pointer Use

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs use of pointer with NULL value or “out of range” value

Where an “out of range” is defined as access to memory chunk thorough exactly one of the following:

- faulty address obtained as a subtraction of two pointers to different memory chunks or
- faulty type such as use of a pointer to access a structure element where the pointer was cast from a data item that is not of a structure datatype

The start statement of the code path is determined by the data flow that associates the pointer to the corresponding data chunk.

Related software fault pattern:

- SFP15 Faulty resource use

The following table lists all discernible CWEs that contribute to this software fault pattern:

476	NULL Pointer Dereference	A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.
469	Use of Pointer Subtraction to Determine Size	The application subtracts one pointer from another in order to determine size, but this calculation can be incorrect if the pointers do not exist in the same memory chunk.
588	Attempt to Access Child of a Non-structure Pointer	Casting a non-structure type to a structure type and accessing a field can lead to memory access errors or data corruption.

4.5.2 Secondary Cluster: Faulty Buffer Access

This cluster covers the common scenarios related to various buffer overflows, underflows and related weaknesses.

This cluster has 11 CWEs. All CWEs in the cluster are discernible.

4.5.2.1 SFP8 Faulty Buffer Access

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs a **Buffer Access Operation** and where exactly one of the following is true:
 - the **access position** of the Buffer Access Operation is outside of the **buffer** or
 - the access position of the Buffer access Operation is inside the buffer and the size of of the data being accessed is greater than the remaining size of the buffer at the access position

This is where the Buffer Access Operation is a statement that performs access to a data item of a certain size at access position. The access position of a Buffer

access Operation is related to a certain buffer and can be either inside the buffer or outside of the buffer.

The following table lists all discernible CWEs that contribute to this software fault pattern:

118	Improper Access of Indexable Resource ('Range Error')	The software does not restrict or incorrectly restricts operations within the boundaries of a resource that is accessed using an index or pointer, such as memory or files.
119	Failure to Constrain Operations within the Bounds of a Memory Buffer	The software may potentially allow operations, such as reading or writing, to be performed at addresses not intended by the developer.
120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
121	Stack-based Buffer Overflow	A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).
122	Heap-based Buffer Overflow	A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().
123	Write-what-where Condition	Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.
124	Boundary Beginning Violation ('Buffer Underwrite')	The software allows a condition where buffers are written to using inappropriate memory access mechanisms such as indexes or pointers that reference memory locations prior to the targeted buffer.
125	Out-of-bounds Read	The software reads data past the end, or before the beginning, of the intended buffer.
126	Buffer Over-read	The software reads data past the end of the intended buffer.
127	Buffer Under-read	The software reads data before the start of the intended buffer.

129	Unchecked Array Indexing	Unchecked array indexing occurs when an unchecked value is used as an index into a buffer.
------------	---------------------------------	--

This software fault pattern has the following parameters:

- Access
 - Reads
 - Writes
- Buffer
 - Stack
 - Heap
- Access Position
 - Array with index
 - Pointer

This cluster has the following parameterization:

- CWE 118: Improper Access of Indexable Resource: An FBA where access position is array with index¹
- CWE 119: Failure to Constrain Operations within the boundaries of a memory buffer: folds under generic FBA
- CWE 121: Stack Overflow: An FBA where the buffer is allocated on the stack and access position is insight the buffer and access is write
- CWE 122: Heap Overflow: An FBA where the buffer is allocated in the heap and access position is insight the buffer and access is write
- CWE 123: Write-what-where Condition: An FBA where access is write
- CWE 124: Buffer Under-write: An FBA where the buffer access is write and access position is outside of the buffer
- CWE 125: Out-of-bounds read: An FBA where the buffer access is read

¹ Here the CWE is constrained to operations on memory buffers.

- CWE 126: Buffer Over-read: An FBA where the buffer access is read and access position is inside of the buffer
- CWE 127: Buffer Under-read: An FBA where the buffer access is read and access position is outside of the buffer
- CWE 129: Unchecked array indexing: An FBA where the access position is array with index

Full parameterization of this SFP is detailed in the accompanying spreadsheet.

4.5.3 Secondary Cluster: Faulty String Expansion

This cluster covers scenarios related to the use of certain API calls that involve implicit buffers and may lead to buffer overflows.

This cluster has 2 discernible CWEs.

4.5.3.1 SFP9 Faulty String Expansion

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that allocates a buffer
- an end statement that performs an implicit buffer access through function call that is characterized by buffer parameters such as the actual buffer size and the expected buffer size where the expected buffer size is greater than the actual buffer size

The following table lists all discernible CWEs that contribute to this software fault pattern:

249	Often Misused: Path Manipulation	Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow.
785	Use of Path Manipulation Function without Maximum-sized Buffer	The software invokes a function for normalizing paths or file names, but it provides an output buffer that is smaller than the maximum possible size, such as <code>PATH_MAX</code> .

4.5.4 Secondary Cluster: Incorrect Buffer Length Computation

This cluster covers scenarios related to several known situations where the length of a buffer is incorrectly computed.

This cluster has 3 CWEs. 2 CWEs in the cluster are discernible. There is 1 non-discernible CWE.

4.5.4.1 SFP10 Incorrect Buffer Length Computation

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs memory allocation for a datatype based on an existing data item of datatype where the computed length of the buffer is incorrect

Where incorrect length of the buffer involves exactly one of the following:

- size of requested buffer is smaller than needs to be
- size of requested buffer is bigger than needs to be

The following table lists all discernible CWEs that contribute to this software fault pattern:

135	Incorrect Calculation of Multi-Byte String Length	The software does not correctly calculate the length of strings that can contain wide or multi-byte characters.
467	Use of sizeof() on a Pointer Type	The code calls sizeof() on a malloced pointer type, which always returns the wordsize/8. This can produce an unexpected result if the programmer intended to determine how much memory has been allocated.

The following table lists non-discernible CWEs in this cluster:

131	Incorrect Calculation of Buffer Size	The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.
------------	--------------------------------------	--

4.5.5 Secondary Cluster: Improper NULL Termination

This cluster covers scenarios related to several operations involving buffer which may lead to buffer overflows due to mismatch in data terminators within the data stored in the buffer.

This cluster has 1 discernible CWE.

4.5.5.1 SFP11 Improper NULL Termination

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that passes a data item to a null-terminated string operation where the data item is non-null-terminated

Where the data can become non-null-terminated in at least one of the following ways:

- data originated from a length-based string operation where the terminator is not automatically added
- null-terminated string was incorrectly transferred and the terminator was omitted
- the null terminator has been overwritten
- array is interpreted as a string where the null terminator is not present in the array

The following table lists all discernible CWEs that contribute to this software fault pattern:

170	Improper Null Termination	The software does not properly terminate a string or array with a null character or equivalent terminator.
------------	---------------------------	--

4.6 Primary Cluster: Memory Management

This is a cluster of weaknesses relates to the management of memory buffers (as opposed to access to memory buffers). Common characteristics of this cluster include:

- Buffer, including stack and heap buffers; static and dynamic buffers
- Buffer identity (pointer, name)
- Buffer allocation operation
- Buffer release operation
- Management of buffer identities

Through these characteristics this cluster is associated with the following clusters:

- Memory access

- Resource management (memory buffer is a special kind of resource)

This cluster contains 5 CWEs. All of the CWEs in this cluster are based on discernible properties and are therefore covered by few software fault patterns.

The Memory Management cluster includes a single secondary cluster:

- Faulty memory release – this cluster covers various scenarios related to incorrect release of memory buffers. The foot-hold of this scenario is the buffer release operation.

4.6.1 Secondary Cluster: Faulty Memory Release

This cluster covers various scenarios related to incorrect release of memory buffers. The foot-hold of this scenario is the buffer release operation.

This cluster has 5 CWEs. All CWEs in the cluster are discernible.

4.6.1.1 SFP12 Faulty Memory Release

Software Fault Pattern - a weakness where the code path has all of the following:

an end statement that releases memory via a reference where the reference points to either incorrect address or incorrect address type NOTE: For example, in C++ some of the valid pairs of allocation/release services are malloc/free, new/delete, new[]/delete[]. Memory buffer allocation should to be supported by “runtime platform knowledge”, which should include all valid pairs for the given platform. The memory buffer status property has to include the type of allocation (including calloc() which involves extra initialization). So, when the pointer is tracked down to the corresponding memory buffer, the type of buffer release is matched to the type of buffer allocation.

The following table lists all discernible CWEs that contribute to this software fault pattern:

415	Double Free	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
590	Free of Memory not on the Heap	The application calls free() on a pointer to memory that was not allocated using associated heap allocation functions such as malloc(), calloc(), or realloc().
761	Free of Pointer not at Start of Buffer	The application calls free() on a pointer to a memory resource that was allocated on the heap, but the pointer is not at the start of the buffer.

762	Mismatched Memory Management Routines	The application attempts to return a memory resource to the system, but it calls a release function that is not compatible with the function that was originally used to allocate that resource.
763	Release of Invalid Pointer or Reference	The application attempts to return a memory resource to the system, but calls the wrong release function or calls the appropriate release function incorrectly.

4.7 Primary Cluster: Resource Management

This cluster of weaknesses relates to management of resources. “Resource” is defined as a dynamic entity provided by the runtime platform. Resources are managed through certain platform-specific APIs. The software application manages the identity of the resource. Common characteristics of this cluster include:

- Resource
- Resource identity
- Management of resource identity
- Resource access operations
- Resource allocation operations
- Resource release operations
- State of resource

Through these characteristics the “Resource Management” cluster is associated with the following other clusters:

- Access control, authentication, privilege (through resource access operations)
- Synchronization (resource can be shared)
- Exception management (through resource state and other status conditions)
- Memory management (memory buffer is a special kind of resource).

This cluster contains 17 CWEs. 12 of the CWEs in this cluster are based on discernible properties and are therefore covered by few software fault patterns. There are 5 non-discernible CWEs in this cluster.

The Resource Management cluster includes the following 4 secondary clusters:

Unrestricted consumption – this cluster covers various scenarios where there is an unrestricted loopback to resource allocation without the corresponding resource release, leading to unlimited consumption of resources, especially in the so-called “request regions”, or the code fragments associated with the user input.

- p. Failure to release resource – this cluster covers various scenarios where the identity of a release is mismanaged resulting in the so-called resource leaks.
- q. Faulty resource use – this cluster covers uses of resource in released (or otherwise incorrect) state
- r. Life cycle – this cluster covers various situations related to incorrect management of resource life cycle. The CWEs in this cluster are all based on non-discernible properties.

4.7.1 Secondary Cluster: Unrestricted Consumption

This cluster covers various scenarios where there is an unrestricted loopback to resource allocation without the corresponding resource release, leading to unlimited consumption of resources, especially in the so-called “request regions”, or the code fragments associated with the user input.

This cluster has 4 CWEs. 3 CWEs in the cluster are discernible. 1 CWE is non-discernible.

4.7.1.1 SFP13 Unrestricted Consumption

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs resource allocation where there is a loopback path and the resource is not released and the allocation control is absent.

This is where “allocation control” is defined as the condition associated with the code path that limits the number of the allocated resource instances.

The following table lists all discernible CWEs that contribute to this software fault pattern:

400	Uncontrolled Resource Consumption ('Resource Exhaustion')	The software does not properly restrict the size or amount of resources that are requested by an actor, which can be used to consume more resources than intended.
674	Uncontrolled Recursion	The product does not properly control the amount of recursion that takes place, which consumes excessive resources, such as allocated memory or

		the program stack.
774	Allocation of File Descriptors or Handles Without Limits or Throttling	The software allocates file descriptors or handles on behalf of an actor without imposing any restrictions on how many descriptors can be allocated, in violation of the intended security policy for that actor.

The following table lists all non-discernible CWEs in this cluster:

770	Allocation of Resources without Limits of Throttling	The software allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on how many resources can be allocated, in violation of the intended security policy for that actor.
------------	--	---

4.7.2 Secondary Cluster: Failure to release resource

This cluster covers various scenarios where the identity of a release is mismanaged resulting in the so-called “resource leaks”.

This cluster has 7 CWEs. All CWEs in the cluster are discernible.

4.7.2.1 SFP14 Failure to release resource

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement performs resource allocation
- an end statement that loses identity of the resource and the resource is not in released state

Where “loses identity” is defined as one of the following:

- resource identity has not been not stored when received
- resource identity has been obtained but was over-written (missing beyond recovery)
- resource identity was never passed to the resource release function
- resource identity is stored in a data item and the data item goes out of scope (no more aliases remain)

- resource identity is stored in a data item and the data item is destroyed.

The following table lists all discernible CWEs that contribute to this software fault pattern:

401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
404	Improper Resource Shutdown or Release	The program does not release or incorrectly releases a resource before it is made available for re-use.
459	Incomplete Cleanup	The software does not properly "clean up" and remove temporary or supporting resources after they have been used.
771	Missing Reference to Active Allocated Resource	The software does not properly maintain a reference to a resource that has been allocated, which prevents the resource from being reclaimed.
772	Missing Release of Resource after Effective Lifetime	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
773	Missing Reference to Active File Descriptor or Handle	The software does not properly maintain references to a file descriptor or handle, which prevents that file descriptor/handle from being reclaimed.
775	Missing Release of File Descriptor or Handle after effective Lifetime	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed.

4.7.3 Secondary Cluster: Faulty Resource Use

This cluster covers uses of resource in released (or otherwise incorrect) state.

This cluster has 2 discernible CWEs.

4.7.3.1 SFP15 Faulty Resource Use

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that performs release of a resource
- an end statement that performs access to a resource and the resource is in released state

NOTE: This software fault pattern is similar to SFP7 Faulty pointer use however this SFP addresses the logical faults that are common to all resources including buffers, while SFP7 addresses specific faults of buffer use.

The following table lists all discernible CWEs that contribute to this software fault pattern:

672	Use of a Resource after Expiration or Release	The software fails to renew or discontinue the use of a resource after expiration, release or revocation.
416	Use After Free	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

4.7.4 Secondary Cluster: Life Cycle

This cluster covers various situations related to incorrect management of resource life cycle. The CWEs in this cluster are all based on non-discernible properties.

Cluster “Life cycle” has 4 non-discernible CWEs.

The following table lists all non-discernible CWEs from this cluster:

664	Improper Control of a Resource Through its Lifetime	The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.
666	Operation on Resource in Wrong Phase of Lifetime	The software performs an operation on a resource at the wrong phase of the resource's lifecycle, which can lead to unexpected behaviors.
675	Duplicate Operations on Resource	The product performs the same operation on a resource two or more times, when the operation should only be applied once.
694	Use of Multiple Resources with Duplicate Identifier	The product uses multiple resources that can have the same identifier, in a context in which unique identifiers are required. This could lead to operations on the wrong resource, or inconsistent operations.

4.8 Primary Cluster: Path Resolution

This cluster of weaknesses relates to access to file resources using complex file names. The weaknesses in this cluster are related to the so-called “path traversal” functionality which is provided by most file systems where the complex file name is interpreted by the

file system using a set of implicit rules. These weaknesses are a common cause of security vulnerabilities. The common characteristics of this cluster include:

- File resources
- File name, including special characters and their interpretation
- File identity
- Chroot jail (the mechanism to restrict interpretation of complex filenames)
- Path equivalence

Through these characteristics the “Path traversal” cluster is associated with the following clusters:

- Tainted input (as the source of the filename or its parts, input transformation, such as canonicalization, etc.)
- Resource management (file resource is a resource)
- Authentication (bypass by alternative name)
- Exception management

There are 51 CWEs in this cluster. 43 CWEs in the cluster are based on discernible properties and are covered by few software fault patterns. There are 8 non-discernible CWEs in this cluster.

The Path Resolution cluster includes the following 3 secondary clusters:

- Path traversal – this cluster covers the majority of patterns leading to path traversal vulnerabilities. The foot-hold of the corresponding software fault pattern is the file access operation where the filename originates from the user input (is “tainted”).
- Failed chroot jail – this cluster covers a specific situation related to incorrect establishment of a chroot jail.
- Link in resource name resolution – this cluster covers situations related to the use of symbolic links to file resources

4.8.1 Secondary Cluster: Path Traversal

This cluster covers the majority of patterns leading to path traversal vulnerabilities. The foot-hold of the corresponding software fault pattern is the file access operation where the filename originates from the user input (is “tainted”).

This cluster has 43 CWEs. 38 CWEs in the cluster are discernible. 5 CWEs are non-discernible.

4.8.1.1 SFP16 Path Traversal

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that accepts input
- an end statement that opens a file using a file path (consisting of a directory name and a filename) where the input is part of the file path and the file path is insecure

Where “insecure file path” is defined as the path of resources that are at least one of the following:

- Resources outside of the access root
- Set of security-sensitive resources

Usually the allowed directory and/or the filename involves one or more special characters that have a special meaning to the runtime platform, such as “/”, “\”, “|”, “\|”, “.”, “..”, “...”, whitespace, etc. Usually the code between the input statement and the end statement involves segments that perform canonicalization and filtering of allowed file paths. The property of “insecure file path” describes various situations where these transformations have been already applied to the user input.

Access to a file outside of the permitted set is an issue if it is not mitigated either by input validation (which restricts the set of file paths) or by establishing a chroot jail (which restricts the way in which the operating system traverses path).

NOTE: the property “access is to a file that is outside of the permitted set” is a complex equation that involves the filename, the current directory and the permitted set, and is formally defined as follows: “path starting at the given current directory allows access to file that is outside of the permitted set”. Properties “access root”, “permitted set of filetypes”, “permitted set of filenames” are often non-discernible, so additional input is required to systematically resolve this equation.

NOTE: this cluster is closely associated with input transformation, such as canonicalization and character encoding, see Primary Cluster Tainted Input. Character encoding and canonicalization may significantly complicate resolving the “path access” equation.

The following table lists all discernible CWEs that contribute to this software fault pattern:

22	Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize special elements that can resolve to a location that is outside of that directory.
23	Relative Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize sequences such as ".." that can resolve to a location that is outside of that directory.
24	Path Traversal: '../filedir'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "../" sequences that can resolve to a location that is outside of that directory.
25	Path Traversal: '/../filedir'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "/../" sequences that can resolve to a location that is outside of that directory.
26	Path Traversal: '/dir/../../filename'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "/dir/../../filename" sequences that can resolve to a location that is outside of that directory.
27	Path Traversal: 'dir/../../filename'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize multiple internal "../" sequences that can resolve to a location that is outside of that directory.
28	Path Traversal: '..\filedir'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "..\" sequences that can resolve to a location that is outside of that directory.
29	Path Traversal: '..\filename'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "..\filename" (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

30	Path Traversal: '\dir\..\filename'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '\dir\..\filename' (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.
31	Path Traversal: 'dir\..\.\filename'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize 'dir\..\.\filename' (multiple internal backslash dot dot) sequences that can resolve to a location that is outside of that directory.
32	Path Traversal: '...' (Triple Dot)	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '...' (triple dot) sequences that can resolve to a location that is outside of that directory.
33	Path Traversal: '....' (Multiple Dot)	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '....' (multiple dot) sequences that can resolve to a location that is outside of that directory.
34	Path Traversal: '....//'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '....//' (doubled dot dot slash) sequences that can resolve to a location that is outside of that directory.
35	Path Traversal: '.../.../'	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '.../.../' (doubled triple dot slash) sequences that can resolve to a location that is outside of that directory.
36	Absolute Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory.
37	Path Traversal: '/absolute/pathname/here'	A software system that accepts input in the form of a slash absolute path

		('/absolute/pathname/here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.
38	Path Traversal: 'absolute\pathname\here'	A software system that accepts input in the form of a backslash absolute path ('absolute\pathname\here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.
39	Path Traversal: 'C:dirname'	An attacker can inject a drive letter or Windows volume letter ('C:dirname') into a software system to potentially redirect access to an unintended location or arbitrary file.
40	Path Traversal: '\\UNC\share\name' (Windows UNC Share)	An attacker can inject a Windows UNC share ('\\UNC\share\name') into a software system to potentially redirect access to an unintended location or arbitrary file.
42	Path Equivalence: 'filename.' (Trailing Dot)	A software system that accepts path input in the form of trailing dot ('filedir.') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	A software system that accepts path input in the form of multiple trailing dot ('filedir....') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
44	Path Equivalence: 'file.name' (Internal Dot)	A software system that accepts path input in the form of internal dot ('file.ordir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
45	Path Equivalence: 'file...name' (Multiple Internal Dot)	A software system that accepts path input in the form of multiple internal dot ('file...dir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

46	Path Equivalence: 'filename ' (Trailing Space)	A software system that accepts path input in the form of trailing space ('filedir ') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
47	Path Equivalence: ' filename (Leading Space)	A software system that accepts path input in the form of leading space (' filedir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
48	Path Equivalence: 'file name' (Internal Whitespace)	A software system that accepts path input in the form of internal space ('file(SPACE)name') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
49	Path Equivalence: 'filename/' (Trailing Slash)	A software system that accepts path input in the form of trailing slash ('filedir/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
50	Path Equivalence: '//multiple/leading/slash'	A software system that accepts path input in the form of multiple leading slash ('//multiple/leading/slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
51	Path Equivalence: '/multiple//internal/slash'	A software system that accepts path input in the form of multiple internal slash ('/multiple//internal/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
52	Path Equivalence: '/multiple/trailing/slash/'	A software system that accepts path input in the form of multiple trailing slash ('/multiple/trailing/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

53	Path Equivalence: <code>'\multiple\\internal\backslash'</code>	A software system that accepts path input in the form of multiple internal backslash (<code>'\multiple\trailing\slash'</code>) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
54	Path Equivalence: <code>'filedir\'</code> (Trailing Backslash)	A software system that accepts path input in the form of trailing backslash (<code>'filedir\'</code>) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
55	Path Equivalence: <code>'./'</code> (Single Dot Directory)	A software system that accepts path input in the form of single dot directory exploit (<code>'./'</code>) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
56	Path Equivalence: <code>'filedir*'</code> (Wildcard)	A software system that accepts path input in the form of asterisk wildcard (<code>'filedir*'</code>) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
57	Path Equivalence: <code>'fakedir../readdir/filename'</code>	The software contains protection mechanisms to restrict access to <code>'readdir/filename'</code> , but it constructs pathnames using external input in the form of <code>'fakedir../readdir/filename'</code> that are not handled by those mechanisms. This allows attackers to perform unauthorized actions against the targeted file.
58	Path Equivalence: Windows 8.3 Filename	The software contains a protection mechanism that restricts access to a long filename on a Windows operating system, but the software does not properly restrict access to the equivalent short "8.3" filename.
67	Improper Handling of Windows Device Names	The software constructs pathnames from user input, but it does not handle or incorrectly handles a pathname containing a Windows device name such as AUX or CON. This typically leads to denial of service or an information leak when the application attempts to process the pathname as a

		regular file.
73	External Control of File Name or Path	The software allows user input to control or influence paths that are used in file system operations.

The following table lists all non-discernible CWEs in this cluster:

41	Improper Resolution of Path Equivalence	The system or application is vulnerable to file system contents disclosure through path equivalence. Path equivalence involves the use of special characters in file and directory names. The associated manipulations are intended to generate multiple names for the same object.
66	Improper Handling of File Names that Identify Virtual Resources	The product does not properly handle a file name that identifies a "virtual" resource that is not directly specified within the directory that is associated with the file name, causing the product to perform file-based operations on a resource that is not a file.
72	Improper Handling of Apple HFS+ Alternate Data Stream Path	The software does not properly handle special paths that may identify the data or resource fork of a file on the HFS+ file system.
428	Unquoted Search Path or Element	The product uses a search path that contains an unquoted element, in which the element contains whitespace or other separators. This can cause the product to access resources in a parent path.
706	Use of Incorrectly-Resolved Name or Reference	The software uses a name or reference to access a resource, but the name/reference resolves to a resource that is outside of the intended control sphere.

4.8.2 Secondary Cluster: Failed Chroot Jail

This cluster covers a specific situation related to incorrect establishment of a chroot jail.

This cluster has 1 discernible CWE.

4.8.2.1 SFP17 Failed Chroot Jail

Software Fault Pattern – a weakness where the code path has all of the following:

- a start statement that has at least one of the following

- performs a chroot or
- performs a chdir; and
- an end statement that opens a file where chroot is activated and the current working directory is outside of the chroot jail

The start statement for the code path is property-driven.

Property: chroot jail status. This property is computed based on the calls to chroot and chdir (as they can occur in any order along the code path). Chroot jail is established when there exists D, such as the chroot call to D has been made and chdir to D has been made, possibly using a relative path. In particular, chdir to D before chroot to D or chdir to “/” after the chroot to D can be made. Chroot jail is failed when a call to chroot has been made, but no call to chdir has been made either before or after the call to chroot. Other more complex situations may be possible, all of which resulting in the situation when the current working directory is outside of the chroot jail. Chroot jail is inactive, when no call to chroot has been made, regardless of calls to chdir.

NOTE: the “chroot jail” is only interesting for a filename injection, where the filename is based on the user input as it attempts to mitigate path resolution issues

NOTE: also need to check that privileges are dropped (setuid is set to non-zero), because chroot can be only executed with root privilege (see related software fault pattern Privilege, CWE 272).

The following table lists all discernible CWEs that contribute to this software fault pattern:

243	Failure to Change Working Directory in chroot Jail	The program uses the chroot() system call to create a jail, but does not change the working directory afterward. This does not prevent access to files outside of the jail.
------------	--	---

4.8.3 Secondary Cluster: Link in Resource Name Resolution

This cluster covers situations related to the use of symbolic links to file resources.

This cluster has 7 CWEs. 4 CWEs in the cluster are discernible. 3 CWEs are non-discernible.

4.8.3.1 SFP18 Link in Resource Name Resolution

Software Fault Pattern - a weakness where the code path has all of the following:

- A start statement that accepts input

- an end statement that opens a file using a file path where the file path is not link-sanitized
- where “no link sanitized” is defined as exactly one of the following:
 - check for link not performed
 - check for link does not cover every segment in the file path

The open statement that occurs on the condition branch of the link check validates against links.

This is supported by a knowledge base facts to find statement that open files and statements that perform link check, as well as how to determine the conditional branch of such statement.

The following table lists all discernible CWEs that contribute to this software fault pattern:

59	Improper Link Resolution Before File Access ('Link Following')	The software attempts to access a file based on the filename, but it does not properly prevent that filename from identifying a link or shortcut that resolves to an unintended resource.
62	UNIX Hard Link	The software, when opening a file or directory, does not sufficiently account for when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.
64	Windows Shortcut Following (.LNK)	The software, when opening a file or directory, does not sufficiently handle when the file is a Windows shortcut (.LNK) whose target is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.
65	Windows Hard Link	The software, when opening a file or directory, does not sufficiently handle when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

The following table lists all non-discernible CWEs in this cluster:

71	Apple '.DS_Store'	Software operating in a MAC OS environment, where .DS_Store is in effect, must carefully manage hard links, otherwise an attacker may be able to leverage a hard link from .DS_Store to overwrite arbitrary files and gain privileges.
386	Symbolic Name not Mapping to Correct Object	A constant symbolic reference to an object is used, even though the reference can resolve to a different object over time.
610	Externally Controlled Reference to a Resource in Another Sphere	The product uses an externally controlled name or reference that resolves to a resource that is outside of the intended control sphere.

4.9 Primary Cluster: Synchronization

This cluster of weaknesses relates to the use of shared resources. Shared resources are accessed by concurrent processes or threads, or any other concurrent computations or external actors. The common characteristics of this cluster include:

- Shared resource and its identity
- Shared resource access
- Locks and their identity
- Critical regions
- Locking issues
- Lock acquisition operations
- Lock release operations
- Guarded regions
- Race condition window

Through these characteristics this cluster is related to the following other clusters:

- Resource management
- Memory access
- Exception management

This cluster contains 22 CWEs. 17 CWEs are described using discernible properties and are covered by few software fault patterns. There are also 4 non-discernible CWEs.

The Synchronization cluster includes the following 4 secondary clusters:

- s. Missing lock – this scenario covers the majority of the synchronization issues where there is no lock guarding a shared resource access (a critical region). This common issue with these situations is the absence of a direct foot-hold.
- t. Race condition window – this cluster cover common scenarios related to sequences of unmediated accesses to shared resource or to externally controlled resources during which the resource can change state. Although common to both “missing lock” and “unchecked status condition” situation, this scenario has a specific foot-hold.
- u. Multiple locks/unlocks - this scenario covers several common locking issues related to the presence of multiple lock operations (lock acquisition or lock release). The foot-hold of this scenario is a lock operation.
- v. Unrestricted lock – this scenario covers a specific locking issue related to a potentially incorrect lock acquisition where the lock is externally controlled and the lock acquisition does not define an alternate flow of control.

4.9.1 Secondary Cluster: Missing Lock

This scenario covers the majority of the synchronization issues where there is no lock guarding a shared resource access (a critical region). This common issue with these situations is the absence of a direct foot-hold.

This cluster has 13 CWEs. 10 CWEs in the cluster are discernible. 3 CWEs are non-discernible.

4.9.1.1 SFP19 Missing lock

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that accesses a shared entity and the entity is improperly synchronized

Where shared entity is exactly one of the following:

- Resource of a particular resource type
- Shared data (including static variables) of a particular resource type.

Where the “improper synchronization” is defined as the situation where there does not exist any lock that synchronizes the shared entity along the given code path or when locks are not adequate.

NOTE: The difference between a resource and shared data is that a resource (a KDM term) is accessed through a platform specific API while shared data is accessed through regular data access actions. A data item can be shared between multiple threads or processes. For this reason we separate memory-related software fault patterns from resource-related software fault patterns. Here the improper synchronization software fault pattern addresses both resources and shared data items because both situations use the same synchronization mechanism.

NOTE: “Race condition in switch” is a trivial case of “Missing lock”, where the shared entity occurs in switch condition.

This software fault pattern is supported by the following knowledge base facts:

<platform> <resource> lock <api> (this is an extension to the “exception management” facts; for locks we need to know the explicit states “locked/unlocked” and “no lock/multiple locks”

The start statement performs locking of the resource (this starts property propagation); this is similar to memory access where we track a certain pointer to a buffer, here however we need to find any lock along the code path as there is no data flow relation between the shared resource and the lock resource; such relation needs to be established via control flow; the lock resource protects the shared resource)

Searching for “no lock” commences from the shared resource and proceeds upwards until the first lock is found, at which point the problem is dismissed. When the end of the code path is reached, the problem is flagged.

The following table lists all discernible CWEs that contribute to this software fault pattern:

364	Signal Handler Race Condition	Race conditions occur frequently in signal handlers, since they are asynchronous actions. These race conditions may have any number of root-causes and symptoms.
365	Race Condition in Switch	The code contains a switch statement in which the switched variable can be modified while the switch is still executing, resulting in unexpected behavior.
366	Race Condition within a Thread	If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.
413	Insufficient Resource Locking	A product does not sufficiently lock resources, in a way that either (1) allows an attacker to simultaneously access those resources, or (2) causes other errors that lead to a resultant weakness.

414	Missing Lock Check	A product does not check to see if a lock is present before performing sensitive operations on a resource.
543	Use of Singleton Pattern in a Non-thread-safe Manner	The use of a singleton pattern may not be thread-safe.
567	Unsynchronized Access to Shared Data	The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.
609	Double-Checked Locking	The program uses double-checked locking to access a resource without the overhead of explicit synchronization, but the locking is insufficient.
662	Insufficient Synchronization	The software attempts to use a shared resource in an exclusive manner, but fails to prevent use by another thread or process.
667	Insufficient Locking	The software does not properly acquire a lock on a resource, leading to unexpected resource state changes and behaviors.

The following table lists the non-discernible CWEs in this cluster:

368	Context Switching Race Condition	A product performs a series of non-atomic actions to switch between contexts that cross privilege or other security boundaries, but a race condition allows an attacker to modify or misrepresent the product's behavior during the switch.
373	State Synchronization Error	State synchronization refers to a set of flaws involving contradictory states of execution in a process which result in undefined behavior.
663	Use of a Non-reentrant Function in an Unsynchronized Context	The software calls a non-reentrant function in a context where a competing thread may have an opportunity to call the same function or otherwise influence its state.

4.9.2 Secondary Cluster: Race Condition Window

This cluster cover common scenarios related to sequences of unmediated accesses to shared resource or to externally controlled resources during which the resource can change state. Although common to both “Missing lock” and “Unchecked status condition”, this scenario has a specific foot-hold.

This cluster has 5 CWEs. 4 CWEs in the cluster are discernible. 1 CWE is non-discernible.

4.9.2.1 SFP20 Race Condition Window

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that checks the status of a resource
- an end statement that performs access to the same resource where the resource access occurs on the conditional branch of start statement and the start statement is not atomic

This weakness identifies a window in the code of a single thread or process where access is made to a resource which is known to be externally accessible by other actors. The state of the resource may change between the two accesses in such a way that the thread finds itself in an unexpected state.

NOTE: related software fault patterns are exception management, resource management and race condition in switch and unconstrained lock

The search commences from the resource access that does not include an atomic status check. We either find a non-atomic status check, in which case we flag an “Improper mediation” problem, or we discover that there is no status check, which should be another problem.

NOTE: CWE does not include entries to flag the missing status check situations.

The following table lists all discernible CWEs that contribute to this software fault pattern:

363	Race Condition Enabling Link Following	The software checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the software to access the wrong file.
367	Time-of-check Time-of-use (TOCTOU) Race Condition	The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.
370	Missing Check for Certificate Revocation after Initial Check	The software does not check the revocation status of a certificate after its initial revocation check, which can cause the software to perform privileged

		actions even after the certificate is revoked at a later time.
638	Failure to Use Complete Mediation	The software does not perform access checks on a resource every time the resource is accessed by an entity, which can create resultant weaknesses if that entity's rights or privileges change over time.

The following table lists all non-discernible CWEs in this cluster:

362	Race Condition	The code requires that certain state should not be modified between two operations, but a timing window exists in which the state can be modified by an unexpected actor or process.
------------	----------------	--

4.9.3 Secondary Cluster: Multiple Locks/Unlocks

This scenario covers several common locking issues related to the presence of multiple lock operations (lock acquisition or lock release). The foot-hold of this scenario is a lock operation.

This cluster has 3 discernible CWE.

4.9.3.1 SFP21 Multiple Locks/Unlocks

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that performs call to change resource's locking state (lock or unlock of a resource)
- an end statement that performs the call of the same locking state that the resource is already in

NOTE1: the "resource is in unlocked state" is defined as follows:

- for a binary lock resource: the lock has not been acquired or the lock has been acquired and has been released
- for a counting lock resource: the lock has not been acquired or the lock has been acquired with a count and has been released count times

the "resource is in locked state" is defined as follows:

- for a binary lock resource: the lock has been acquired and has not been released

- for a counting lock resource: the lock has been acquired with a count and has been released less than count times

NOTE2: CWE does not have an entry which describes a situation where there is a missing unlock. This is partly covered by a multiple lock situation, where there is a loopback path to the original lock statement after the access to the shared resource (the critical section). This can be described as: L=lock->critical section->L

Multiple locks also describe a different case: LL=lock->lock->critical section and its variations.

However this does not cover a trivial case of: LS=lock-> critical section-> stop (without a loopback).

Searching for multiple locks on a shared resource requires a different search: not only do we need to find the first lock going up from the shared resource, but the second, etc. Therefore multiple locks is defined as a separate software fault pattern, where the end statement is a lock; the search then becomes the same as for “no lock” – we need to find the first lock to flag the problem, or to hit the end of the code path to dismiss the problem. When we hit the “unlock” we also dismiss the problem and stop the search.

A lock synchronizes a resource (such as a shared resource or another lock) if the resource access occurs on the synchronized branch of the lock.

The following table lists all discernible CWEs that contribute to this software fault pattern:

764	Multiple Locks of a Critical Resource	The software locks a critical resource more times than intended, leading to an unexpected state in the system.
765	Multiple Unlocks of a Critical Resource	The software unlocks a critical resource more times than intended, leading to an unexpected state in the system.
585	Empty Synchronized Block	The software contains an empty synchronized block.

4.9.4 Secondary Cluster: Unrestricted Lock

This scenario covers a specific locking issue related to a potentially incorrect lock acquisition where the lock is externally controlled and the lock acquisition does not define an alternate flow of control.

This cluster has 1 discernible CWE.

4.9.4.1 SFP22 Unrestricted Lock

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs lock of a resource and the resource is externally accessible and there is no alternative flow (the flow will be stuck if the resource becomes locked externally)

The following table lists all discernible CWEs in this cluster:

412	Unrestricted Lock on Critical Resource	The software properly checks for the existence of a lock on a critical resource, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.
-----	--	---

4.10 Primary Cluster: Information Leak

This cluster of weaknesses relates to the export of sensitive information from an application and several related issues. The common characteristics of this cluster include:

- Sensitive data (defined as data which flow from sensitive operations or flows into sensitive operations as the key parameter. “Sensitive” is the role that a data element plays in a certain context. We can know this role based on the APIs that are involved in producing/consuming/transforming the data element. If a data element was passed to a password management function, it can be assumed to be a password. If a data element is passed to a function that is known to require a private key – it is a private key.)
- Information export operations (including storing, logging, releasing as an error message, releasing as a debug message, as well as other exposures)
- State disclosure
- Temporary file and their names
- Data in motion
- Data at rest and their configuration
- Output channels
- Buffer cleanup

Through these characteristics this cluster is associated with the following other clusters:

- Memory management
- Channel (output channel)

- Synchronization
- Resource management
- Exception management

This cluster contains 96 CWEs. Most CWEs are described using non-discernible properties, so only 37 contribute to software fault patterns. There are 57 non-discernible CWEs in this cluster.

The Information Leak cluster includes the following 5 secondary clusters:

- Exposed data - this cluster covers various situations related to the data motion, data at rest etc., which leads to information leaks, where there is corresponding code with sufficient foot-hold for a white-box description
- Insecure session management– this cluster covers several scenarios related to information leaks between sessions; CWEs in this cluster do not have sufficient white-box content
- Other exposures – this cluster covers various miscellaneous scenarios leading to information leak, not covered by the previous clusters. CWEs in this cluster do not have sufficient white-box content.
- State disclosure – this cluster covers various situations of state disclosure, which releases the knowledge of some aspects of the internal state of the application. CWEs in this cluster do not have sufficient white-box content
- Exposure through temporary files - this cluster covers scenarios related to temporary files management, in particular to their names. CWEs in this cluster do not have sufficient white-box content

4.10.1 Secondary Cluster Exposed Data

This cluster contains 76 CWEs. Most CWEs are described using non-discernible properties, so only 38 contribute to software fault patterns. There are 38 non-discernible CWEs in this cluster.

This cluster is further subdivided into the following 8 groups:

- Exposed data in motion - this group covers various situations related to the data motion, which leads to information leaks, where there is corresponding code with sufficient foot-hold for a white-box description

- Exposure through storing – this group covers various situations related to the data at rest, which leads to information leaks, where there is corresponding code with sufficient foot-hold for a white-box description
- Exposed data at rest - this group covers various situations related to the data at rest, which leads to information leaks, as there is no corresponding code or no sufficient foot-hold for a white-box description
- Exposure through logging - this group covers various situations related to the data in use, which leads to information leaks through logging, where there is corresponding code with sufficient foot-hold for a white-box description
- Exposure through debug message - this group covers various situations related to the data in use, which leads to information leaks through debug messages, where there is corresponding code with sufficient foot-hold for a white-box description
- Exposure through error message- this group covers various situations related to the data in use, which leads to information leaks through error messages, where there is corresponding code with sufficient foot-hold for a white-box description
- Inappropriate cleanup – this group covers several buffer cleanup weaknesses
- Programmatic exposures of data – this group covers several scenarios related to miscellaneous constructs leading to information release

4.10.1.1 SFP23 Exposed Data

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement performs moving data where the data is sensitive and the data is inadequately protected

Where “inadequately protected data” is defined as exactly one of the following:

- Data that is not encrypted (in cleartext, in plaintext)
- data that is not sanitized

Where “sensitive data” is defined as used in APIs that are intended for handling sensitive data (for example passwords API)

4.10.1.2 Exposed Data in Motion

This group defines various explicit “data transmitting” operations that result in releasing information; each operation corresponds to an API call with the data value being released as the key parameter.

This group has 8 CWEs. 3 CWEs in the group are discernible. 5 CWEs are non-discernible.

The following table lists all discernible CWEs in this group:

311	Failure to Encrypt Sensitive Data	The failure to encrypt data passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.
319	Cleartext Transmission of Sensitive Information	The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.
523	Unprotected Transport of Credentials	Login pages not using adequate measures to protect the user name and password while they are in transit from the client to the server.

The following table lists all non-discernible CWEs in this group:

5	J2EE Misconfiguration: Data Transmission Without Encryption	Information sent over a network can be compromised while in transit. An attacker may be able to read/modify the contents if the data are sent in plaintext or are weakly encrypted.
200	Information leak (Information Disclosure)	An Information leak is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.
201	Information leak Through Sent Data	The accidental leaking of sensitive information through sent data refers to the transmission of data which are either sensitive in and of itself or useful in the further exploitation of the system through standard data channels.
212	Cross-boundary Cleansing Information leak	The software does not properly remove sensitive data from a source when preparing it for, or transferring it to, an untrusted destination.
213	Intended Information leak	A product's design or configuration explicitly requires the publication of information that could be regarded as sensitive by an administrator.

NOTE: CWE 5 and 319 are very similar: both make a claim about plaintext/cleartext; 5 makes additional claim about weak encryption; 5 mentions “a network that can be compromised” while 319 mentions “communication channel that can be sniffed by unauthorized actors.”; 5 mentions “information”, while 319 mentions “sensitive or security-critical data”

201 is also similar to 5 and 319: it mentions “sensitive information”

212 is a different facet of the same situation: “does not properly remove sensitive information when transferring to an untrusted party”.

4.10.13 Exposure Through Storing

This group defines various situations related to the data at rest of the software system, where explicit “data storing” operations are involved; each operation correspond to an API call with the data value being released as the key parameter.

This group has 13 CWEs. There are 8 discernible CWEs in this group and 5 non-discernible ones.

The following table lists all discernible CWEs in this group:

256	Plaintext Storage of a Password	Storing a password in plaintext may result in a system compromise.
257	Storing Passwords in a Recoverable Format	The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. If a system administrator can recover a password directly, or use a brute force search on the available information, the administrator can use the password on other accounts.
312	Cleartext Storage of Sensitive Information	The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere, when the information should be encrypted or otherwise protected.
313	Plaintext Storage in a File or on Disk	Storing sensitive data in plaintext in a file, or on disk, makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.
314	Plaintext Storage in the Registry	Storing sensitive data in plaintext in the registry makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

315	Plaintext Storage in a Cookie	Storing sensitive data in plaintext in a cookie makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.
317	Plaintext Storage in GUI	Storing sensitive data in plaintext within the GUI makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.
642	External Control of Critical State Data	The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors.

The following table lists all non-discernible CWEs in this group:

13	ASP.NET Misconfiguration: Password in Configuration File	Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource making them an easy target for attackers.
260	Password in Configuration File	The software stores a password in a configuration file that might be accessible to actors who do not know the password.
522	Insufficiently Protected Credentials	This weakness occurs when the application transmits or stores authentication credentials and uses an insecure method that is susceptible to unauthorized interception and/or retrieval.
539	Information leak Through Persistent Cookies	Persistent cookies are cookies that are stored on the browser's hard drive. This can cause security and privacy issues depending on the information stored in the cookie and how it is accessed.
555	J2EE Misconfiguration: Plaintext Password in Configuration File	The J2EE application stores a plaintext password in a configuration file.

4.10.1.4 Exposed Data at Rest

This group covers various situations related to data at rest of the software system which leads to information leaks, as there is no corresponding code or no sufficient foot-hold for a white-box description.

This group has 19 CWEs. 3 CWEs in the group are discernible. 16 CWEs are non-discernible.

The following table lists all discernible CWEs that contribute to this group:

533	Information leak Through Server Log Files	A server.log file was found. This can give information on whatever application left the file. Usually this can give full path names and system information, and sometimes usernames and passwords.
534	Information leak Through Debug Log Files	The application does not sufficiently restrict access to a log file that is used for debugging.
542	Information leak Through Cleanup Log Files	The application fails to protect or delete a log file related to cleanup.

The following table lists all non-discernible CWEs in this group:

219	Sensitive Data Under Web Root	The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties.
220	Sensitive Data Under FTP Root	The application stores sensitive data under the FTP document root with insufficient access control, which might make it accessible to untrusted parties.
318	Plaintext Storage in Executable	Sensitive information should not be stored in plaintext in an executable. Attackers can reverse engineer a binary code to obtain secret data.
433	Unparsed Raw Web Content Delivery	The software stores raw content or supporting code under the web document root with an extension that is not specifically handled by the server, resulting in an Information leak.
527	Information leak Through CVS Repository	Information contained within a CVS directory left as a subdirectory on a webserver (such as usernames, filenames, path root and IP addresses) could be recovered by an attacker and used for malicious purposes.
528	Information leak Through Core Dump Files	The application generates a core dump file in a directory that is accessible to parties outside of the intended control sphere.

529	Information leak Through Access Control List Files	These files allow the attacker to know the setup of the security Access Control Lists. This will give the attacker information that may allow the attacker to bypass the security of the site.
530	Information leak Through Backup (.~bk) Files	Often, old files are renamed with an extension such as .~bk to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved.
538	File and Directory Information leaks	Weaknesses in this category are related to Information leaks in files and directories.
540	Information leak Through Source Code	Source code on a web server often contains sensitive information and should generally not be accessible to users.
541	Information leak Through Include Source Code	If an include file source is accessible, the file can contain usernames and passwords, as well as sensitive information pertaining to the application and system.
546	Suspicious Comment	The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.
548	Information leak Through Directory Listing	A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.
552	Files or Directories Accessible to External Parties	Files or directories are accessible in the environment that should not be.
612	Information leak Through Indexing of Private Data	The product performs an indexing routine against private documents, but does not sufficiently verify that the actors who can access the index also have the privileges to access the private documents.
615	Information leak Through Comments	While adding general comments is very useful, some programmers tend to leave important data, such as: filenames related to the web application, old links or links which were not meant to be browsed by users, old code fragments, etc.

4.10.1.5 Exposure Through Logging

This group defines various explicit “logging” operations that result in releasing information; each operation correspond to an API call with the data value being released as the key parameter.

This group has 2 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs in this group:

117	Improper Output Sanitization for Logs	The software does not properly sanitize or incorrectly sanitizes output that is written to logs.
532	Information leak Through Log Files	Information written to log files can be of a sensitive nature and give valuable guidance to an attacker.

4.10.1.6 Exposure Through Debug Message

This group defines various scenarios that result in releasing information through debug messages; where each debug message is released by a specific operation corresponding to an API call with the data value being released as the key parameter.

This group has 3 CWEs. 2 CWEs in the group are discernible. There is 1 non-discernible CWE in this group.

The following table lists all discernible CWEs in this group:

215	Information leak Through Debug Information	The application contains debugging code that can leak sensitive information to untrusted parties.
497	Information leak of System Data	Revealing system data or debugging information helps an adversary learn about the system and form an attack plan.

The following table lists all non-discernible CWEs in this group:

11	ASP.NET Misconfiguration: Creating Debug Binary	Debugging messages help attackers learn about the system and plan a form of attack.
-----------	---	---

4.10.1.7 Exposure Through Error Message

This group defines various scenarios that result in releasing information through error messages; where each error message is released by a specific operation corresponding to an API call with the data value being released as the key parameter.

This group has 10 CWEs. 2 CWEs in the group are discernible. 8 CWEs are non-discernible.

The following table lists all discernible CWEs in this group:

209	Error Message Information leak	The software generates an error message that includes sensitive information about its environment, users, or associated data.
210	Product-Generated Error Message Information leak	The software identifies an error condition and creates its own diagnostic or error messages that contain sensitive information.

The following table lists all non-discernible CWEs in this group:

7	J2EE Misconfiguration: Missing Custom Error Page	The default error page of a web application should not display sensitive information about the software system.
12	ASP.NET Misconfiguration: Missing Custom Error Page	An ASP .NET application must enable custom error pages in order to prevent attackers from mining information from the framework's built-in responses.
211	Product-External Error Message Information leak	The software performs an operation that triggers an external diagnostic or error message that is not directly generated by the software, such as an error generated by the programming language interpreter that the software uses. The error can contain sensitive system information.
535	Information leak Through Shell Error Message	A command shell error message indicates that there exists an unhandled exception in the web application code. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.
536	Information leak Through Servlet Runtime Error Message	A servlet error message indicates that there exists an unhandled exception in your web application code and may provide useful information to an attacker.
537	Information leak Through Java Runtime Error Message	In many cases, an attacker can leverage the conditions that cause unhandled exception errors in order to gain unauthorized access to the system.
550	Information leak Through Server Error Message	Certain conditions, such as network failure, will cause a server error message to be displayed.

756	Missing Custom Error Page	The software fails to return custom error pages to the user, possibly resulting in an information leak.
------------	---------------------------	---

4.10.1.8 Programmatic Exposures of Data

This group covers several scenarios related to miscellaneous constructs leading to information release.

This group has 16 CWEs. 13 CWEs in the group are discernible. 3 CWEs are non-discernible.

The following table lists all discernible CWEs that contribute to this group:

8	J2EE Misconfiguration: Entity Bean Declared Remote	When an application exposes a remote interface for an entity bean, it might also expose methods that get or set the bean's data. These methods could be leveraged to read sensitive information, or to change data in ways that violate the application's expectations, potentially leading to other vulnerabilities.
214	Process Environment Information leak	A process is invoked with sensitive arguments, environment variables, or other elements that can be seen by other processes on the operating system.
316	Plaintext Storage in Memory	Storing sensitive data in plaintext in memory makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.
403	UNIX File Descriptor Leak	A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.
495	Private Array-Typed Field Returned From A Public Method	The product has a method that is declared public, but returns a reference to a private array, which could then be modified in unexpected ways.
498	Information leak through Class Cloning	The code contains a class with sensitive data, but the class is cloneable. The data can then be accessed by cloning the class.
499	Serializable Class Containing Sensitive Data	The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class

		through another class.
501	Trust Boundary Violation	The product mixes trusted and untrusted data in the same data structure or structured message.
526	Information leak Through Environmental Variables	Environmental variables may contain sensitive information about a remote server.
591	Sensitive Data Storage in Improperly Locked Memory	The application stores sensitive data in memory that is not locked, or that has been incorrectly locked, which might cause the memory to be written to swap files on disk by the virtual memory manager. This can make the data more accessible to external actors.
598	Information leak Through Query Strings in GET Request	The web application uses the GET method to process requests that contain sensitive information, which can expose that information through the browser's history, Referers, web logs, and other sources.
607	Public Static Final Field References Mutable Object	A public or protected static final field references a mutable object, which allows the object to be changed by malicious code, or accidentally from another package
767	Access to Critical Private Variable via Public Method	The software defines a public method that reads or modifies a private variable.
374	Mutable Objects Passed by Reference	Sending non-cloned mutable data as an argument may result in that data being altered or deleted by the called function, thereby putting the calling function into an undefined state.
375	Passing Mutable Objects to an Untrusted Method	Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the calling function, thereby putting the class in an undefined state.

The following table lists all non-discernible CWEs in this group:

402	Transmission of Private Resources into a New Sphere ('Resource Leak')	The software makes resources available to untrusted parties when those resources are only intended to be accessed by the software.
668	Exposure of Resource to Wrong Sphere	The product exposes a resource to the wrong sphere, in ways that are not related to incorrectly specified permissions.

669	Incorrect Resource Transfer Between Spheres	The product does not properly transfer a resource/behavior to another sphere, or improperly imports a resource/behavior from another sphere, in a manner that provides unintended control over that resource.
------------	---	---

4.10.1.9 Inappropriate Cleanup

This group covers several buffer cleanup weaknesses

This group has 3 CWEs. All CWEs in the group are discernible.

The following table lists all discernible CWEs that contribute to this group:

14	Compiler Removal of Code to Clear Buffers	Sensitive memory is cleared according to the source code, but compiler optimizations leave the memory untouched when it is not read from again, aka "dead store removal."
226	Sensitive Information Uncleared Before Release	The software does not fully clear previously used information in a data structure, file, or other resource, before making that resource available to a party in another control sphere.
244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	Using realloc() to resize buffers that store sensitive information can leave the sensitive information exposed to attack, because it is not removed from memory.

NOTE: This group is related to SFP14 Failure to release resource (CWE 459) which addresses the situation of resource release and Cluster Incorrect Exception Behavior (CWE 460).

4.10.2 Secondary Cluster: State Disclosure

This cluster covers various situations of state disclosure, which releases the knowledge of some aspects of the internal state of the application. CWEs in this cluster do not have sufficient white-box content.

This cluster has 7 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

202	Privacy Leak through Data Queries	When trying to keep information confidential, an attacker can often infer some of the information by using statistics.
------------	-----------------------------------	--

203	Discrepancy Information leaks	A discrepancy Information leak is an Information leak in which the product behaves differently, or sends different responses, in a way that reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.
204	Response Discrepancy Information leak	The software provides different responses to incoming requests in a way that allows an actor to determine system state information that is outside of that actor's control sphere.
205	Behavioral Discrepancy Information leak	A behavioral discrepancy Information leak occurs when the product's actions indicate important differences based on (1) the internal state of the product or (2) differences from other products in the same class.
206	Internal Behavioral Inconsistency Information leak	Two separate operations in a product cause the product to behave differently in a way that is observable to an attacker and reveals security-relevant information about the internal state of the product, such as whether a particular operation was successful or not.
207	External Behavioral Inconsistency Information leak	The software behaves differently than other products like it, in a way that is observable to an attacker and reveals security-relevant information about which product is being used, or its operating state.
208	Timing Discrepancy Information leak	Two separate operations in a product require different amounts of time to complete, in a way that is observable to an actor and reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.

4.10.3 Secondary Cluster: Exposure Through Temporary files

This cluster covers scenarios related to temporary files management, in particular to their names. CWEs in this cluster do not have sufficient white-box content.

This cluster has 3 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

377	Insecure Temporary File	Creating and using insecure temporary files can leave application and system data vulnerable to attack.
378	Creation of Temporary File With Insecure Permissions	Opening temporary files without appropriate measures or controls can leave the file, its contents and any function that it impacts vulnerable to attack.
379	Creation of Temporary File in Directory with Incorrect Permissions	The software creates a temporary file in a directory whose permissions allow unintended actors to determine the file's existence or otherwise access that file.

4.10.4 Secondary Cluster: Other Exposures

This cluster covers various miscellaneous scenarios leading to information leak, not covered by the previous clusters. CWEs in this cluster do not have sufficient white-box content.

This cluster has 7 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

453	Insecure Default Variable Initialization	The software, by default, initializes an internal variable with an insecure or less secure value than is possible.
485	Insufficient Encapsulation	The product does not sufficiently encapsulate critical data or functionality.
487	Reliance on Package-level Scope	Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.
492	Use of Inner Class Containing Sensitive Data	Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.
525	Information leak Through Browser Caching	For each web page, the application should have an appropriate caching policy specifying the extent to which the page and its form fields should be cached.
614	Sensitive Cookie in HTTPS Session Without "Secure" Attribute	The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over

		an HTTP session.
651	Information leak through WSDL File	The Web services architecture may require exposing a WSDL file that contains information on the publicly accessible services and how callers of these services should interact with them (e.g. what parameters they expect and what types they return).

4.10.5 Secondary Cluster: Insecure Session Management

This cluster covers several scenarios related to information leaks between sessions; CWEs in this cluster do not have sufficient white-box content.

This cluster has 3 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

6	J2EE Misconfiguration: Insufficient Session-ID Length	The J2EE application is configured to use an insufficient session ID length.
488	Data Leak Between Sessions	The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.
524	Information leak Through Caching	The application uses a cache to maintain a pool of objects, threads, connections, pages, or passwords to minimize the time it takes to access them or the resources to which they connect. If implemented improperly, these caches can allow access to unauthorized information or cause a denial of service vulnerability.

4.11 Primary Cluster: Tainted Input

This cluster groups weaknesses related to injection of user controlled data into various destination commands. This cluster focuses at the data validation issues. The common characteristics of this cluster include:

- Destination command or construct
- Data validation, special characters and their interpretation

- Tainted values
- Channel (input channel)
- Input transformation (encoding, canonicalization, etc.)
- Input handling (processing complex input structures)

Through these characteristics, the Tainted Input cluster is associated to the following other clusters:

- Risky values, Memory access (through various properties of a memory buffer which may be destinations of the tainted values)
- Path resolution (through data validation and tainted values)
- Resource management
- Authentication (through input handling)
- Protocol errors
- Exception management

This cluster contains 138 CWEs. 79 CWEs contribute to software fault patterns. 59 CWEs are non-discernible.

The Tainted Input cluster includes the following 6 secondary clusters:

- Tainted input to command – this cluster covers various scenarios that involve data validation and in particular the special characters for various destinations commands
- Tainted input to variable - this cluster covers scenarios where the destination of the tainted values is not an API call, but some construct, for example, a basic condition, a loop condition, etc.
- Tainted input to environment – this cluster covers scenarios where the tainted values affect various element of the computation environment which has an indirect effect on the computation itself
- Faulty input transformation – this cluster covers several scenarios related to the transformation of input, such as encoding, canonicalization, etc.
- Incorrect input handling – this cluster covers several scenarios related to processing of complex input structures

- Composite tainted input – this cluster is introduced to describe vulnerabilities in which user controlled input contributes to other weaknesses, for example a buffer overflow in which the buffer length is tainted data

Tainted Input secondary clusters that have discernible properties consisting of 3 types: Tainted Input to Command (TIC) Type, Tainted Input to Environment (TIE), and Tainted Input to Variable (TIV).

The following sections describe the secondary clusters of “Tainted Input” and their current status in relation to CWEs.

4.11.1 Secondary Cluster: Tainted Input to Command

This cluster covers various scenarios that involve data validation and in particular the special characters for various destinations commands.

This cluster has 87 CWEs. 68 CWEs in the cluster are discernible. 19 CWEs are non-discernible.

Characteristics of Tainted Input to Command (TIC):

- Is related to the input data validation driven by syntax of the command that input data contributes to and is being interpreted and executed by the platform.
“Command” is defined as a certain API call.
- Some examples of CWEs are injection data (e.g. OS injection, SQL injection, Resource injection) and XSS
- Currently some of CWEs covering this space have 2 flaws:
 - defined input data validation is mostly independent from the destination command data that input contributes to.
 - number of CWEs are focused on use of validation interface within particular frameworks which besides describing a good practice has no impact on assessment of input validation effectiveness (e.g., validation can be very poor (non-effective) using recommended interface or very good but bypassing interface)

The following tables list parameters/data elements found in CWEs that are related to condition that forms tainted data weakness. These elements are input commands (Table 3), destination commands (Table 4) and special characters/symbols (Table 5).

Table 3. Input Commands of TIC Type

Input command/component entry point
Struts and input validation framework
HTTP request
ASP.NET and input validation framework
PHP request
Upstream Component

Table 4. Destination Commands of TIC Type

Destination command
HTTP
SQL
OS
LDAP
XML
SMTP
EVAL

Table 5. Special Characters of TIC Type

Special char/sym
& < >
* () [] ; , . : \$

CRLF
“”
delimiters
•
Record delimiters
Line delimiters
Section delimiters
Expression/command delimiters
Escape, Meta, control sequences
Comment delimiters
Macro symbols
Substitution characters
Variable name delimiters
Wildcard or matching symbols
whitespace
Paired delimiters
NULL
Multiple leading special elements
Multiple trailing special elements
Missing/additional/inconsistent special element

Table 6 lists CWE IDs in relationship to parameters/data elements (what CWE focuses and reports)

Table 6. CWEs in Relationship to Parameters of TIC Type

Row ID	Parameters ----- CWE ID	Input Cmd	Use of Validation Interface	Destination Command	Validation: Special Characters & Symbols	Validation: Applicable Design Specification
1	93; 138; 140; 141; 142; 143; 144; 145; 146; 147; 148; 149; 150; 151; 152; 153; 154; 155; 156; 157; 161; 162; 163; 164; 165; 641;				Y	
2	102; 103; 108; 104; 105; 109; 110; 554;		Y			
3	77; 78; 79; 80; 113; 84; 86; 112; 601; 644	Y		Y	Y (incomplete list)	
4	90; 624; 74; 81; 82; 85; 87; 134; 564; 89; 91; 611; 619; 643; 652; 77; 78; 95; 96;	Implied		Y	Mentioned but not listed	
5	15; 20; 99 (resource); 566; 621; 641;	Implied		Implied		Y
6	83;	Y			Mentioned but not listed	

Notes:

- CWEs identified in Row 1 are of no value since special characters are not provided in the context of destination command that are intended for
- CWEs identified in row 2 are of no value since they are focused on recommendation/best practice to enable framework validation interface – that still does not tell us how good validation is or if it exist only that is enabled

- The only valid CWEs are those in rows 3, 4 and 5; however for completeness they need to be expended to include list of categorized destination commands with prohibited special characters

4.11.1.1 SFP24 Tainted Input to Command

Tainted Input to Command (TIC):

- Software Fault Pattern - A weakness where the code path has all of the following:
 - a start statement that accepts input data
 - an end statement that executes destination command where the input data is part of **destination command** and the input data is undesirable

Where “input is undesirable” is defined exactly one of the following:

- not validated
- incorrectly validated against **special characters and symbols** that trigger certain functionality during execution of **destination command** (discernible) and against **applicable design specification**

Example of concrete parameters for a parameterized TIC SFP is shown in Table 7.

Table 7. Example: Concrete Parameters for Parameterized TIC SFP

Input Command / Component Entry Point		Destination		
		Destination Command	Validation: Special Characters and Symbols	Validation: applicable design specification
HTTP	Any Input can be connected to any Destination Command	SQL command	1. Specifically: NUL (0x00) BS (0x08) TAB (0x09) LF (0x0a) CR (0x0d) SUB (0x1a) " (0x22) % (0x25) ' (0x27) \ (0x5c) _ (0x5f) 2. Good practice: All other non- alphanumeric characters within ASCII set of characters	not necessary
Command Line Interface		HTTP output stream, e.g., OutputStream::Write	< > " ' % ;) (& + -	not necessary
Input Files, Environment. variables input...		OS Command, e.g., java.lang.Runtime System.Diagnostics.Proc ess.Start exec() or passthru()	It is more platform specific, however if we want to be generic, we can say: All non-alphanumeric characters within ASCII set of characters	not necessary
Socket Read		Command accessing any resource, e.g., socket, file	N/A	Valid resource id
RMI (as example for component entry point)		Command that accepts format string, e.g., Printf();	No validation is good for these commands – they should be flagged regardless of validation	

The following table lists all discernible CWEs that contribute to this software fault pattern:

74	Failure to Sanitize Data into a Different Plane ('Injection')	The software fails to adequately filter user-controlled input data for syntax that has control-plane implications.
77	Failure to Sanitize Data into a Control Plane ('Command Injection')	The software fails to adequately filter command (control plane) syntax from user-controlled input (data plane) and then allows potentially injected commands to execute within its context.
78	Failure to Preserve OS Command Structure ('OS Command Injection')	The software uses externally-supplied input to dynamically construct all or part of a command, which is then passed to the operating system for execution, but the software does not sufficiently enforce which commands and arguments are specified.
79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	The software does not sufficiently validate, filter, escape, and encode user-controllable input before it is placed in output that is used as a web page that is served to other users.
80	Improper Sanitization of Script-Related HTML Tags in a Web Page (Basic XSS)	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special characters such as "<", ">", and "&" that could be interpreted as web-scripting elements when they are sent to a downstream component that processes web pages.
81	Improper Sanitization of Script in an Error Message Web Page	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special characters that could be interpreted as web-scripting elements when they are sent to an error page.
82	Improper Sanitization of Script in Attributes of IMG Tags in a Web Page	The web application does not filter or incorrectly filters scripting elements within attributes of HTML IMG tags, such as the src attribute.
83	Failure to Sanitize Script in Attributes in a Web Page	The software does not filter "javascript:" or other URI's from dangerous attributes within tags, such as onmouseover, onload, onerror, or style.
84	Failure to Resolve Encoded URI Schemes in a Web Page	The web application fails to filter user-controlled input for executable script disguised with URI encodings.
85	Doubled Character XSS	The web application fails to filter user-controlled

	Manipulations	input for executable script disguised using doubling of the involved characters.
86	Failure to Sanitize Invalid Characters in Identifiers in Web Pages	The software does not strip out invalid characters in the middle of tag names, URI schemes, and other identifiers, which are still rendered by some web browsers that ignore the characters.
87	Failure to Sanitize Alternate XSS Syntax	The software fails to adequately filter user-controlled input for alternate script syntax.
89	Failure to Preserve SQL Query Structure ('SQL Injection')	The application dynamically generates an SQL query based on user input, but it does not sufficiently prevent that input from modifying the intended structure of the query.
90	Failure to Sanitize Data into LDAP Queries ('LDAP Injection')	The software does not sufficiently sanitize special elements that are used in LDAP queries or responses, allowing attackers to modify the syntax, contents, or commands of the LDAP query before it is executed.
91	XML Injection (aka Blind XPath Injection)	The software does not properly filter or quote special characters or reserved words that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.
93	Failure to Sanitize CRLF Sequences ('CRLF Injection')	The software uses CRLF (carriage return line feeds) as a special element, e.g. to separate lines or records, but it does not properly sanitize CRLF sequences from inputs.
95	Improper Sanitization of Directives in Dynamically Evaluated Code ('Eval Injection')	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes code syntax before using the input in a dynamic evaluation call (e.g. "eval").
96	Improper Sanitization of Directives in Statically Saved Code ('Static Code Injection')	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes code syntax before inserting the input into an executable resource, such as a library, configuration file, or template.
99	Improper Sanitization of Resource Identifiers ('Resource Injection')	The software receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.

102	Struts: Duplicate Validation Forms	The application uses multiple validation forms with the same name, which might cause the Struts Validator to validate a form that the programmer does not expect.
103	Struts: Incomplete validate() Method Definition	The application has a validator form that either fails to define a validate() method, or defines a validate() method but fails to call super.validate().
104	Struts: Form Bean Does Not Extend Validation Class	If a form bean does not extend an ActionForm subclass of the Validator framework, it can expose the application to other weaknesses related to insufficient input validation.
105	Struts: Form Field Without Validator	The application has a form field that is not validated by a corresponding validation form, which can introduce other weaknesses related to insufficient input validation.
108	Struts: Unvalidated Action Form	Every Action Form must have a corresponding validation form.
109	Struts: Validator Turned Off	Automatic filtering via a Struts bean has been turned off, which disables the Struts Validator and custom validation logic. This exposes the application to other weaknesses related to insufficient input validation.
110	Struts: Validator Without Form Field	Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.
112	Missing XML Validation	Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input.
113	Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	The software fails to adequately filter HTTP headers for CR and LF characters.
130	Improper handling of Length Parameter Inconsistency	The software does not handle or incorrectly handles incoming data that contains a length or size field that is inconsistent with the actual length of the associated data.
134	Uncontrolled Format String	The software uses externally-controlled format strings in printf-style functions, which can lead to buffer overflows or data representation problems.

138	Improper Sanitization of Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as control elements when they are sent to a downstream component.
140	Failure to Sanitize Delimiters	The software does not properly sanitize delimiters.
141	Failure to Sanitize Parameter/Argument Delimiters	Parameter delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.
142	Failure to Sanitize Value Delimiters	Value delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.
143	Failure to Sanitize Record Delimiters	Record delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.
144	Failure to Sanitize Line Delimiters	Line delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.
145	Failure to Sanitize Section Delimiters	Section delimiters injected into an application can be used to compromise a system.
146	Failure to Sanitize Expression/Command Delimiters	Delimiters between expressions or commands injected into the software through input can be used to compromise a system.
147	Improper Sanitization of Input Terminators	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as input terminators when they are sent to a downstream component.
148	Failure to Sanitize Input Leaders	The application does not properly handle when a leading character or sequence ("leader") is missing or malformed, or if multiple leaders are used when only one should be allowed.
149	Failure to Sanitize Quoting Syntax	Quotes injected into an application can be used to compromise a system. As data are parsed, an injected/absent/duplicate/malformed use of quotes

		may cause the process to take unexpected actions.
150	Failure to Sanitize Escape, Meta, or Control Sequences	Escape, meta, or control character/sequence injected into an application through input can be used to compromise a system.
151	Improper Sanitization of Comment Delimiters	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as comment delimiters when they are sent to a downstream component.
152	Improper Sanitization of Macro Symbols	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as macro symbols when they are sent to a downstream component.
153	Improper Sanitization of Substitution Characters	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as substitution characters when they are sent to a downstream component.
154	Improper Sanitization of Variable Name Delimiters	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as variable name delimiters when they are sent to a downstream component.
155	Improper Sanitization of Wildcard or Matching Symbols	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as wildcards or matching symbols when they are sent to a downstream component.
156	Improper Sanitization of Whitespace	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as whitespace when they are sent to a downstream component.
157	Failure to Sanitize Paired Delimiters	The software does not properly handle the characters that are used to mark the beginning and ending of a group of entities, such as parentheses, brackets, and braces.
158	Failure to Sanitize Null Byte or NUL Character	NUL characters or null bytes injected into an application through input can be used to compromise a system.

161	Improper Sanitization of Multiple Leading Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes multiple leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.
163	Improper Sanitization of Multiple Trailing Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes multiple trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.
165	Improper Sanitization of Multiple Internal Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes multiple internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.
554	ASP.NET Misconfiguration: Not Using Input Validation Framework	The ASP.NET application does not use an input validation framework.
564	SQL Injection: Hibernate	Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.
566	Access Control Bypass Through User-Controlled SQL Primary Key	Without proper access control, executing a SQL statement that contains a user-controlled primary key can allow an attacker to view unauthorized records.
601	URL Redirection to Untrusted Site ('Open Redirect')	A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. This simplifies phishing attacks.
611	Information leak Through XML External Entity File Disclosure	The product processes an XML document that can contain XML entities with URLs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.
619	Dangling Database Cursor ('Cursor Injection')	If a database cursor is not closed properly, then it could become accessible to other users while retaining the same privileges that were originally assigned, leaving the cursor "dangling."

621	Variable Extraction Error	The product uses external input to determine the names of variables into which information is extracted, without verifying that the names of the specified variables are valid. This could cause the program to overwrite unintended variables.
624	Executable Regular Expression Error	The product uses a regular expression that either (1) contains an executable component with user-controlled inputs, or (2) allows a user to enable execution by inserting pattern modifiers.
641	Insufficient Filtering of File and Other Resource Names for Executable Content	When an application does not restrict the valid names of resources (e.g. files) supplied by the user, various problems may arise down the line when these resources are used.
643	Failure to Sanitize Data within XPath Expressions ('XPath injection')	The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not sufficiently sanitize that input. This allows an attacker to control the structure of the query.
644	Improper Sanitization of HTTP Headers for Scripting Syntax	The application does not sanitize or incorrectly sanitizes web scripting syntax in HTTP headers that can be used by web browser components that can process raw headers, such as Flash.
652	Failure to Sanitize Data within XQuery Expressions ('XQuery Injection')	The software uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not sufficiently sanitize that input. This allows an attacker to control the structure of the query.

The following table lists all non-discernible CWEs in this cluster:

75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	The software fails to adequately filter user-controlled input for special elements with control implications.
76	Failure to Resolve Equivalent Special Elements into a Different Plane	The software fails to adequately filter non-typical special elements that are equivalent to control-relevant special elements that are already being filtered.
88	Argument Injection or Modification	The software does not sufficiently delimit the arguments being passed to a component in another

		control sphere, allowing alternate arguments to be provided, leading to potentially security-relevant changes.
92	Improper Sanitization of Custom Special Characters	The software uses a custom or proprietary language or representation, but when it receives input from an upstream component, it does not sanitize or incorrectly sanitizes special elements when they are sent to a downstream component.
97	Failure to Sanitize Server-Side Includes (SSI) Within a Web Page	The software fails to adequately filter server-side include (control-plane) syntax from user-controlled input (data plane) and then allows potentially injected server-side includes to be acted upon.
100	Technology-Specific Input Validation Problems	Weaknesses in this category are caused by inadequately implemented input validation within particular technologies.
106	Struts: Plug-in Framework not in Use	When an application does not use an input validation framework such as the Struts Validator, there is a greater risk of introducing weaknesses related to insufficient input validation.
107	Struts: Unused Validation Form	An unused validation form indicates that validation logic is not up-to-date.
159	Failure to Sanitize Special Element	Weaknesses in this attack-focused category fail to sufficiently filter and interpret special elements in user-controlled input which could cause adverse effect on the software behavior and integrity.
160	Improper Sanitization of Leading Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.
162	Improper Sanitization of Trailing Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.
164	Improper Sanitization of Internal Special Elements	The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

183	Permissive Whitelist	An application uses a "whitelist" of acceptable values, but the whitelist includes at least one unsafe value, leading to resultant weaknesses.
184	Incomplete Blacklist	An application uses a "blacklist" of prohibited values, but the blacklist is incomplete.
185	Incorrect Regular Expression	The software specifies a regular expression in a way that causes data to be improperly sanitized or compared.
186	Overly Restrictive Regular Expression	A regular expression is overly restrictive, which prevents dangerous values from being detected.
444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	When malformed or abnormal HTTP requests are interpreted by one or more entities in the data flow between the user and the web server, such as a proxy or firewall, they can be interpreted inconsistently, allowing the attacker to "smuggle" a request to one device without the other device being aware of it.
553	Command Shell in Externally Accessible Directory	A possible shell file exists in /cgi-bin/ or other accessible directories. This is extremely dangerous and can be used by an attacker to execute commands on the web server.
625	Permissive Regular Expression	The product uses a regular expression that does not sufficiently restrict the set of allowed values.
626	Null Byte Interaction Error (Poison Null Byte)	The product does not properly handle null bytes or NUL characters when passing data between different representations or components.
627	Dynamic Variable Evaluation	In a language where the user can influence the name of a variable at runtime, if the variable names are not controlled, an attacker can read or write to arbitrary variables, or access arbitrary functions.
646	Reliance on File Name or Extension of Externally-Supplied File	The software allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.
707	Improper Enforcement of Messages or Data Structure	The software does not enforce or incorrectly enforces that structured messages or data are well-formed before being read from an upstream component or sent to a downstream component.

4.11.2 Secondary Cluster: Tainted Input to Variable

This cluster covers scenarios where the destination of the tainted values is not an API call, but a programmatic construct, for example, a basic condition, a loop condition, etc. This Software Fault Pattern covers situations where the tainted input flows into a variable thus creating a possibility for the attacker to exercise certain control the computation. This Software Fault Pattern uses a distinct pattern in which the foot-hold is the usage of a data element.

4.11.2.1 SFP25 Tainted Input to Variable

Software Fault Pattern - a weakness where the code path has all of the following:

- The end statement that uses tainted data value.

Where tainted data value is defined as externally controlled value obtained through at least one of the following scenarios:

- Not validated user input
- Not validated configuration settings
- Not validated environment variables

Where tainted data value use is defined by exactly one the following scenarios:

- Used as a program's control variable
- Assigned to a variable.

Where "user input" is defined as data originating through a certain platform-specific resource which is accessed using a system call

This cluster has 8 CWEs. All CWEs in the cluster are discernible.

The following table lists all discernible CWEs that contribute to this software fault pattern:

15	External Control of System or Configuration Setting	One or more system settings or configuration elements can be externally controlled by a user.
20	Improper Input Validation	The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.
454	External Initialization of Trusted Variables	The software initializes critical internal variables using inputs that can come from externally controlled sources.

606	Unchecked Input for Loop Condition	The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service because of excessive looping.
496	Public Data Assigned to Private Array-Typed Field	Assigning public data to a private array is equivalent to giving public access to the array.
502	Deserialization of Untrusted Data	The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
616	Incomplete Identification of Uploaded File Variables (PHP)	The PHP application uses an old method for processing uploaded files by referencing the four global variables that are set for each file (e.g. \$varname, \$varname_size, \$varname_name, \$varname_type). These variables could be overwritten by attackers, causing the application to process unauthorized files.

4.11.3 Secondary Cluster: Composite Tainted Input

This cluster is introduced to describe vulnerabilities in which user controlled input contributes to other weaknesses, for example a buffer overflow in which the buffer length is tainted data.

There are CWEs that are currently described using this approach.

Characteristics of Composite tainted input

- Is related to the input data validation flowing into an existing weakness condition or weakness code path that input data contributes to
- allows composite descriptions of vulnerabilities where user controlled input contributed to other weaknesses and turns them into exploitable security holes
- Currently no CWEs are described as Composite tainted input pattern. This is why buffer overflow weaknesses do not mention user controlled data as properties of the buffer access operation. This allows better descriptions of vulnerabilities and their associated weaknesses.

4.11.3.1 SFP26 Composite Tainted Data

Software Fault Pattern – a weakness where the code path has all of the following:

- a start statement that accepts input data through exactly one of the following:
 - a) executing "input command" or
 - b) component entry point (API that can be invoked by platform)

- an end statement that is an end statement of another SFP that uses a data value where the input data contributes to the data value
- the condition of the SFP of the end statement is satisfied

NOTE: this software fault pattern describes composition of an existing weakness and how one of its key data values is affected by user-controlled input data. This SFP defines the origin of input data. The end statement (destination of data) and the condition are defined in separate SFPs. Currently several SFP explicitly include "Origin of input data" parameter, which refers to SFP26. However, the Composite SFP approach can be used to describe multiple situations where the un-validated user input controls another weakness.

4.11.4 Secondary Cluster: Faulty input Transformation

This cluster covers several scenarios related to the transformation of input, such as encoding, canonicalization, etc.

This cluster has 15 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

116	Improper Encoding or Escaping of Output	The software prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structure of the message is not preserved.
166	Improper Handling of Missing Special Element	The software receives input from an upstream component, but it does not handle or incorrectly handles when an expected special element is missing.
167	Improper Handling of Additional Special Element	The software receives input from an upstream component, but it does not handle or incorrectly handles when an additional unexpected special element is missing.
168	Failure to Resolve Inconsistent Special Elements	The software does not handle when an inconsistency exists between two or more special characters or reserved words.
172	Encoding Error	The software fails to properly handle encoding or decoding of the data, resulting in unexpected values.
173	Failure to Handle Alternate Encoding	The software does not properly handle when an input uses an alternate encoding that is valid for the control sphere to which the input is being sent.

174	Double Decoding of the Same Data	The software decodes the same input twice, which can limit the effectiveness of any protection mechanism that occurs in between the decoding operations.
175	Failure to Handle Mixed Encoding	The software does not properly handle when the same input uses several different (mixed) encodings.
176	Failure to Handle Unicode Encoding	The software does not properly handle when an input contains Unicode encoding.
177	Failure to Handle URL Encoding (Hex Encoding)	The software does not properly handle when all or part of an input has been URL encoded.
178	Failure to Resolve Case Sensitivity	The software does not properly account for differences in case sensitivity when accessing or determining the properties of a resource, leading to inconsistent results.
179	Incorrect Behavior Order: Early Validation	The software validates input before applying protection mechanisms that modify the input, which could allow an attacker to bypass the validation via dangerous inputs that only arise after the modification.
180	Incorrect Behavior Order: Validate Before Canonicalize	The software validates input before it is canonicalized, which prevents the software from detecting data that becomes invalid after the canonicalization step.
181	Incorrect Behavior Order: Validate Before Filter	The software validates data before it has been filtered or cleansed, which prevents the software from detecting data that becomes invalid after the filtering step.
182	Collapse of Data Into Unsafe Value	The software cleanses or filters data in a way that causes the data to be reduced or "collapsed" into an unsafe value.

4.11.5 Secondary Cluster: Incorrect Input Handling

This cluster covers several scenarios related to processing of complex input structures. This cluster has 17 non-discernible CWEs.

The following table lists all non-discernible CWEs in this cluster:

198	Use of Incorrect Byte Ordering	The software receives input from an upstream component, but it does not account for byte ordering (e.g., big-endian and little-endian) when processing the input, causing an incorrect number or value to be used.
228	Improper Handling of Syntactically Invalid Structure	The product does not handle or incorrectly handles input that is not syntactically well-formed with respect to the associated specification.
229	Improper Handling of Values	Weaknesses in this category are related to missing or incorrect handling of values that are associated with parameters, fields, or arguments.
230	Improper Handling of Missing Values	The software does not handle or incorrectly handles when a parameter, field, or argument name is specified, but the associated value is missing, i.e. it is empty, blank, or null.
231	Improper Handling of Extra Values	The software does not handle or incorrectly handles when more values are specified than expected.
232	Improper Handling of Undefined Values	The software does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.
233	Parameter Problems	Weaknesses in this category are related to improper handling of parameters, fields, or arguments.
234	Failure to Handle Missing Parameter	If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.
235	Improper Handling of Extra Parameters	The software does not handle or incorrectly handles when a particular parameter, field, or argument name is specified two or more times.
236	Improper Handling of Undefined Parameters	The software does not handle or incorrectly handles when a particular parameter, field, or argument name is not defined or supported by the product.
237	Improper Handling of Structural Elements	The software does not handle or incorrectly handles inputs that are related to complex structures..

238	Improper Handling of Incomplete Structural Elements	The application does not handle or incorrectly handles when a particular structural element is not completely specified.
239	Failure to Handle Incomplete Element	The application does not properly handle when a particular element is not completely specified.
240	Improper Handling of Inconsistent Structural Elements	The software does not handle or incorrectly handles when two or more structural elements should be consistent, but are not.
241	Improper Handling of Unexpected Data Type	The application does not handle or incorrectly handles when a particular element is not the expected type, e.g. it expects a digit (0-9) but is provided with a letter (A-Z).
351	Insufficient Type Distinction	The software does not properly distinguish between different types of elements in a way that leads to insecure behavior.
354	Improper Validation of Integrity Check Value	The software does not validate or incorrectly validates the integrity check values or "checksums" of a message. This may prevent it from detecting if the data has been modified or corrupted in transmission.

4.11.6 Secondary Cluster: Tainted Input to Environment

This cluster covers scenarios where the tainted values affect various element of the computation environment which has an indirect effect on the computation itself.

This cluster has 11 CWEs. 3 CWEs in the cluster are discernible. There are also 8 non-discernible CWEs.

4.11.6.1 SFP27 Tainted Input to Environment

Software Fault Pattern - a weakness where the code path has all of the following:

- The end statement that calls a tainted control element.

Where tainted control element is defined by exactly one the following scenarios:

- Loading library from untrusted sourceDynamically loading code from untrusted source
- External function hook

The following table lists all discernible CWEs that contribute to this software fault pattern:

494	Download of Code Without Integrity Check	The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.
545	Use of Dynamic Class Loading	Dynamically loaded code has the potential to be malicious.
622	Unvalidated Function Hook Arguments	A product adds hooks to user-accessible API functions, but does not properly validate the arguments. This could lead to resultant vulnerabilities.

The following table lists all non-discernible CWEs:

94	Failure to Control Generation of Code ('Code Injection')	The product does not sufficiently filter code (control-plane) syntax from user-controlled input (data plane) when that input is used within code that the product generates.
114	Process Control	Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker.
427	Uncontrolled Search Path Element	One or more locations in a static search path are under control of the attacker.
470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	The application uses external input with reflection to select which classes or code to use, but it does not sufficiently prevent the input from selecting improper classes or code.
471	Modification of Assumed-Immutable Data (MAID)	The software does not properly protect an assumed-immutable element from being modified by an attacker.
472	External Control of Assumed-Immutable Web Parameter	The web application does not sufficiently verify inputs that are assumed to be immutable but are actually externally controllable, such as hidden form fields.
473	PHP External Variable Modification	A PHP application does not properly protect against the modification of variables from external sources, such as query parameters or cookies. This

		can expose the application to numerous weaknesses that would not exist otherwise.
673	External Influence of Sphere Definition	The product does not prevent the definition of control spheres from external actors.

4.12 Primary Cluster: Entry Points

This cluster of weaknesses relates to unexpected entry points into the application. Extra entry points may be defined by leftover debug code or testing code. The extra entry points are associated with extra debug computations or test computations, which may contain various related weaknesses, not contained in the production computation. The common characteristics of this cluster include:

- Component entry point
- Production computation
- Test computation
- Debug computation

Through these characteristics this cluster is associated to the following clusters:

- Authentication
- Information leak
- Malware
- Channel
- UI

This cluster contains 11 CWEs. All CWEs are described using discernible properties and contribute to a software fault pattern.

This cluster has only one secondary cluster (no further differentiation).

4.12.1 Secondary Cluster: Unexpected Access Points

This cluster has 11 CWEs. All CWEs in the cluster are discernible.

4.12.1.1 SFP28 Unexpected Access Points

Software Fault Pattern - a weakness where the code path has all of the following:

- Statement that defines an entry point into an application where the entry point is not required by production code.

Where “entry point that is not required by production code” is defined as any functionality not required by the production code such as at least one of the following:

- Debug code, or
- Test code, or
- Access to data

NOTE: We focus at the entry point itself rather than at the information or sensitive information that is exposed by the code associated with the entry point.

The following table lists all discernible CWEs that contribute to this software fault pattern:

489	Leftover Debug Code	The application can be deployed with active debugging code that can create unintended entry points.
531	Information leak Through Test Code	Accessible test applications can pose a variety of security risks. Since developers or administrators rarely consider that someone besides themselves would even know about the existence of these applications, it is common for them to contain sensitive information or functions.
608	Struts: Non-private Field in ActionForm Class	An ActionForm class contains a field that has not been declared private, which can be accessed without using a setter or getter.
491	Public cloneable() Method Without Final ('Object Hijack')	A class has a cloneable() method that is not declared final, which allows an object to be created without calling the constructor. This can cause the object to be in an unexpected state.
493	Critical Public Variable Without Final Modifier	The product has a critical public variable that is not final, which allows the variable to be modified to contain unexpected values.
500	Public Static Field Not Marked Final	An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.
568	finalize() Method Without super.finalize()	The software contains a finalize() method that does not call super.finalize().
580	clone() Method Without super.clone()	The software contains a clone() method that fails to call super.clone() to obtain the new object.

582	Array Declared Public, Final, and Static	The program declares an array public, final, and static, which is not sufficient to prevent the array's contents from being modified.
583	finalize() Method Declared Public	The program violates secure coding principles for mobile code by declaring a finalize() method public.
766	Critical Variable Declared Public	The software declares a critical variable or field to be public when intended security policy requires it to be private.

4.13 Primary Cluster: Authentication

This cluster of weaknesses relates to establishing the identity of an actor associated with the computation, or the identity of the endpoint involved in the computation through a certain channel. Authentication cluster is closely related to Access Control which focuses at resource access by an authenticated actor with appropriate access rights as well as ownership of the resources by the authenticated actors.

The common characteristics of the “Authentication” cluster include the following:

- Authentication mechanism, including password
- Authenticated actor, its identity and management
- Authentication check
- Management of actors
- Sensitive data

Through these characteristics the “Authentication” cluster is associated with the following clusters:

- Access control
- Entry point
- UI
- Information leak
- Resource access
- Tainted input (through input processing)
- Exception management

This cluster contains 43 CWEs. Only 14 CWEs contribute to software fault patterns. 29 CWEs are described using non-discernible properties. The major challenge of the “Authentication” cluster is the lack of good foot-holds for the white-box description. In particular the “Authentication” concept is itself the purpose of a certain code region rather than a white-box pattern. Authentication often involves complex logic.

The “Authentication” cluster includes the following 9 secondary clusters:

- Authentication bypass – this cluster covers situations related to incomplete authentication steps; there is no sufficient white-box content in this cluster
- Faulty endpoint authentication – this cluster covers scenarios involved in endpoint authentication; the foot-hold in this scenario is a certain condition which uses an inappropriate authentication mechanism
- Missing endpoint authentication – this cluster covers scenarios where the endpoint authentication is absent. The foot-hold of this scenario is the resource access or a critical operation.
- Digital certificate – this cluster covers specific authentication issues related to digital certificate management.
- Missing authentication – this cluster covers scenarios where the authentication is absent and the resource access or critical operation occur at a code region where the corresponding actor is not authenticated
- Insecure authentication policy – this cluster covers miscellaneous policy issues related to authentication. There is no sufficient white-box content in the CWEs in this cluster
- Multiple binds to the same port – this cluster covers a specific pattern describing multiple binds to the same port.
- Hardcoded sensitive data – this cluster covers various situations where the sensitive data involved in authentication checks is hardcoded.
- Unrestricted authentication – this cluster covers specific situation where there is a loopback in the unauthenticated region, leading back to the authentication, without sufficient control

4.13.1 Secondary Cluster: Authentication Bypass

This cluster covers situations related to incomplete authentication steps; there is no sufficient white-box content in this cluster.

This cluster has 10 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

287	Improper Authentication	When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.
288	Authentication Bypass Using an Alternate Path or Channel	A product requires authentication, but the product has an alternate path or channel that does not require authentication.
289	Authentication Bypass by Alternate Name	The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.
303	Incorrect Implementation of Authentication Algorithm	The requirements for the software dictate the use of an established authentication algorithm, but the implementation of the algorithm is incorrect.
304	Missing Critical Step in Authentication	The software implements an authentication technique, but it skips a step that weakens the technique.
305	Authentication Bypass by Primary Weakness	The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication error.
308	Use of Single-factor Authentication	The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.
309	Use of Password System for Primary Authentication	The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.
592	Authentication Bypass Issues	The software does not properly perform authentication, allowing it to be bypassed through various methods.
603	Use of Client-Side Authentication	A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check.

4.13.2 Secondary Cluster: Faulty Endpoint Authentication

This cluster covers scenarios involved in endpoint authentication; the foot-hold in this scenario is a certain condition which uses an inappropriate authentication mechanism.

This cluster has 11 CWEs. 6 CWEs in the cluster are discernible.

4.13.2.1 SFP29 Faulty Endpoint Authentication

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that performs input with the input data
- an end statement that performs condition check that involves data value where the data value is a property of the input data and the property provides insufficient endpoint authentication

Where “property of the input data” is defined as the result of the API calls marked in the knowledge base as returning properties of the input data where the input data is passed as the key parameter to the call. There can be chains of such properties, for example `getRemoteAddr()->getByName()->getCanonicalHostName()`, where the first call operates on a request and the last call return a property of that request.

Where “property provides insufficient endpoint authentication” is defined a property that is marked in the knowledge base as insufficient for authentication.

NOTE: the end statement occur at the “request region” of the input, where the “request region” is defined as the code fragment that is reachable from the input by control flow. This is already incorporated in the definition of a code path, in which the start statement performs input – the end statement by definition occurs at the “request region” on the input.

The following table lists all discernible CWEs that contribute to this software fault pattern:

247	Reliance on DNS Lookups in a Security Decision	Attackers can spoof DNS entries. Do not rely on DNS names for security.
292	Trusting Self-reported DNS Name	The use of self-reported DNS names as authentication is flawed and can easily be spoofed by malicious users.
293	Using Referer Field for Authentication	The referer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

350	Improperly Trusted Reverse DNS	The software trusts the hostname that is provided when performing a reverse DNS resolution on an IP address, without also performing forward resolution.
360	Trust of System Event Data	Security based on event locations is insecure and can be spoofed.
565	Use of Cookies in Security Decision	The software relies on the existence or values of cookies when making a security decision.

The following table lists all non-discernible CWEs in this cluster:

302	Authentication Bypass by Assumed-Immutable Data	The authentication scheme or implementation uses key data elements that are assumed to be immutable, but can be controlled or modified by the attacker.
345	Insufficient Verification of Data Authenticity	The software does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.
346	Origin Validation Error	The software does not properly verify that the source of data or communication is valid.
551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	If a web server does not fully parse requested URLs before it examines them for authorization, it may be possible for an attacker to bypass authorization protection.
647	Use of Non-Canonical URL Paths for Authorization Decisions	The software defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical. This can allow a non-canonical URL to bypass the authorization.

4.13.3 Secondary Cluster: Missing Endpoint Authentication

This cluster covers scenarios where the endpoint authentication is absent. The foot-hold of this scenario is the resource access or a critical operation.

This cluster has 2 CWEs. All CWEs in the cluster are discernible.

4.13.3.1 SFP30 Missing Endpoint Authentication

Software Fault Pattern - a weakness where the code path has all of the following:

- a start statement that performs input with input data

- an end statement that accesses the input data where the end statement occurs at the region that is not guarded by endpoint authentication conditions of the input data

Where “endpoint authentication condition of an input data” is a condition that involves data value that is a property of the input data.

Where “statement accesses input data” is defined as an API call which is defined in the knowledge base as accessing the input data and where the input data is a passed as the key parameter to the call.

Where a “region guarded by condition” is a code fragment that starts at the selected branch of the condition (the branch where the condition is considered validated).

The following table lists all discernible CWEs that contribute to this software fault pattern:

422	Unprotected Windows Messaging Channel ('Shatter')	The software does not properly verify the source of a message in the Windows Messaging System while running at elevated privileges, creating an alternate channel through which an attacker can directly send a message to the product.
425	Direct Request ('Forced Browsing')	The web application fails to adequately enforce appropriate authorization on all restricted URLs, scripts or files.

4.13.4 Secondary Cluster: Digital Certificate

This cluster covers specific authentication issues related to digital certificate management.

This cluster has 6 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

296	Improper Following of Chain of Trust in Certificate Validation	The chain of trust is not followed or is incorrectly followed when validating a certificate, resulting in incorrect trust of any resource that is associated with that certificate.
297	Improper Validation of Host-specific Certificate Data	Host-specific certificate data is not validated or is incorrectly validated, so while the certificate read is valid, it may not be for the site originally

		requested.
298	Improper Validation of Certificate Expiration	A certificate expiration is not validated or is incorrectly validated, so trust may be assigned to certificates that have been abandoned due to age.
299	Improper Check for Certificate Revocation	The software does not check or incorrectly checks the revocation status of a certificate, which may cause it to use a certificate that has been compromised.
593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	The software modifies the SSL context after connection creation has begun.
599	Trust of OpenSSL Certificate Without Validation	The failure to validate certificate data may mean that an attacker may be claiming to be a host which it is not.

4.13.5 Secondary Cluster: Missing Authentication

This cluster covers scenarios where the authentication is absent and the resource access or critical operation occurs at a code region where the corresponding actor is not authenticated.

This cluster has 2 CWEs. All CWEs in the cluster are discernible.

4.13.5.1 SFP31 Missing Authentication

Software Fault Pattern - a weakness where the code path has all of the following

- an end statement that performs a critical operation where the end statement occurs at the region that is not guarded by authentication conditions

Where “authentication condition” is a condition that involves a data value that is a result of an API call that is marked in the knowledge base as authentication call.

Where “statement performs critical operation” is defined as an API call which is defined in the knowledge base as performing a critical operation (for example, a password change).

The following table lists all discernible CWEs in this cluster:

306	No Authentication for Critical Function	The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.
------------	---	--

620	Unverified Password Change	When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.
------------	----------------------------	---

4.13.6 Secondary Cluster: Insecure Authentication Policy

This cluster covers miscellaneous policy issues related to authentication. There is no sufficient white-box content in the CWEs in this cluster.

This cluster has 6 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

262	Not Using Password Aging	If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner.
263	Password Aging with Long Expiration	Allowing password aging to occur unchecked can result in the possibility of diminished password integrity.
521	Weak Password Requirements	The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.
556	ASP.NET Misconfiguration: Use of Identity Impersonation	Configuring an ASP.NET application to run with impersonated credentials may give the application unnecessary privileges.
613	Insufficient Session Expiration	According to WASC, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization."
645	Overly Restrictive Account Lockout Mechanism	The software contains an account lockout protection mechanism, but the mechanism is too restrictive and can be triggered too easily. This allows attackers to deny service to legitimate users by causing their accounts to be locked out.

4.13.7 Secondary Cluster: Multiple binds to the Same Port

This cluster covers a specific pattern describing multiple binds to the same port.

This cluster has 1 CWE. All CWEs in the cluster are discernible.

4.13.7.1 SFP32 Multiple Binds to the Same Port

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs binding of a socket where the socket allows multiple bindings

Where “socket allows multiple bindings” is defined as a specific value “bind to all interfaces” INADDR_ANY or 0.0.0.0.

The following table lists all discernible CWEs in this cluster:

605	Multiple Binds to the Same Port	When multiple sockets are allowed to bind to the same port, other services on that port may be stolen or spoofed.
------------	---------------------------------	---

4.13.8 Secondary Cluster: Hardcoded Sensitive Data

This cluster covers various situations where the sensitive data involved in authentication checks is hardcoded. Sensitive data is defined as data which flow from sensitive operations or flows into sensitive operations as the key parameter. “Sensitive” is the role that a data element plays in a certain context. We can know this role based on the APIs that are involved in producing/consuming/transforming the data element. If a data element was passed to a password management function, it can be assumed to be a password. If a data element is passed to a function that is known to require a private key – it is a private key. The fact that a certain string is a “password” or a “private key” is a role that the string plays in some context. This role can be in some cases detected by examining the APIs involved.

The runtime knowledge base should distinguish between the roles of a “public key” and the “private key”. It’s a hard-coded crypto key used for encryption that is the problem. A hard-coded “public key”, used for authentication/identification has less impact. Ideally, it wouldn’t be COMPLETELY hard-coded, so that you could remove a bad key, but these aren’t the weakness that a hard-coded ENCRYPTION key is.

This cluster has 4 CWEs. 2 CWEs in the cluster are discernible. 2 CWEs are non-discernible.

4.13.8.1 SFP33 Hardcoded Sensitive Data

Software Fault Pattern – a weakness where the code path has all of the following:

- a start statement that accepts **sensitive data item**
- an end statement that performs comparison where one operand is the **sensitive data item** and the other operand is a constant value

The following table lists all discernible CWEs that contribute to this software fault pattern:

259	Hard-Coded Password	The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.
321	Use of Hard-coded Cryptographic Key	The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.

The following table lists all non-discernible CWEs in this cluster:

258	Empty Password in Configuration File	Using an empty string as a password is insecure.
547	Use of Hard-coded, Security-relevant Constants	The program uses hard-coded constants instead of symbolic names for security-critical values, which increases the likelihood of mistakes during code maintenance or security policy change.

4.13.9 Secondary Cluster: Unrestricted Authentication

This cluster covers specific situation where there is a loopback in the unauthenticated region, leading back to the authentication, without sufficient restriction.

This cluster has 1 discernible CWE.

4.13.9.1 SFP34 Unrestricted Authentication

Software Fault Pattern - a weakness where the code path has all of the following:

- an end statement that performs authentication where the unvalidated branch has a loopback path and the authentication control is absent

Where “authentication control” is defined as the condition associated with the code path that limits the number of the authentication attempts.

Where a “statement that performs authentication” is an authentication condition

The following table lists all discernible CWEs that contribute to this software fault pattern:

307	Failure to Restrict Excessive Authentication Attempts	The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.
------------	---	---

4.14 Primary Cluster: Access Control

This cluster of weaknesses relates to validating resource owners and their permissions. The common characteristics of this cluster include:

- Authenticated actor, its identity and management
- Access rights and their management
- Resource, protected resource
- Resource ownership
- Access control check, protected region
- Resource access operation
- Operation that sets access rights on a resource
- Privilege
- Sensitive data

Through these characteristics the “Access control” cluster is associated to the following clusters:

- Authentication
- Privilege
- Resource management
- Exception management
- Information leak (in particular though insecure permanent data store)

There are 16 CWEs in this cluster. Most CWEs in this cluster are described using non-discernible properties.

The Access Control cluster includes the following 3 secondary clusters:

- Insecure resource access – this cluster covers situations related to the bypass of access control checks
- Access management – this cluster covers various situations related to the management of resource owners and access rights. There is no sufficient white-box content in CWE descriptions for this cluster
- Insecure resource permissions – this cluster covers various scenarios related to setting permissions of the resources. The foot-hold of this scenario is the operation that sets resource permissions (such as resource creation, cloning, or explicit permission setting)

4.14.1 Secondary Cluster: Insecure Resource Access

This cluster covers situations related to the bypass of access control checks.

This cluster has 4 CWEs. 2 CWEs in the cluster are discernible. 2 CWEs are non-discernible.

4.14.1.1 SFP35 Insecure Resource Access

Software Fault Pattern - a weakness where the code path has all of the following:

- An End statement that performs a resource access to a resource where the end statement occurs at the region that is not guarded by access control conditions involving the resource.

Where “access control condition involving a resource” is a condition that involves data value that is a result of an API call that is marked in the knowledge base as authentication call where the API call is passed another data value that is a property of the resource

Where “statement performs resource access” is defined as an API call which is defined in the knowledge base as performing a resource access

Where “property of resource” is defined as the result of the API calls marked in the knowledge base as returning properties of the resource where the resource identity is passed as the key parameter to the call. There can be chains of such properties.

The following table lists all discernible CWEs that contribute to this software fault pattern:

285	Improper Access Control (Authorization)	The software does not perform or incorrectly performs access control checks across all potential execution paths.
424	Failure to Protect Alternate Path	The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

The following table lists all non-discernible CWEs in this cluster:

639	Access Control Bypass Through User-Controlled Key	The system's access control functionality does not prevent one user from gaining access to another user's records by modifying the key value identifying the record.
650	Trusting HTTP Permission Methods on the Server Side	The server contains a protection mechanism that assumes that any URI that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass intended access restrictions and conduct resource modification and deletion attacks, since some applications allow GET to modify state.

4.14.2 Secondary Cluster: Insecure Resource Permissions

This cluster covers various scenarios related to setting permissions of the resources. The foot-hold of this scenario is the operation that sets resource permissions (such as resource creation, cloning, or explicit permission setting).

This cluster has 7 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

276	Incorrect Default Permissions	The software, upon installation, sets incorrect permissions for an object that exposes it to an unintended actor.
277	Insecure Inherited Permissions	A product defines a set of insecure permissions that are inherited by objects that are created by the program.
278	Insecure Preserved Inherited Permissions	A product inherits a set of insecure permissions for an object, e.g. when copying from an archive file, without user awareness or involvement.

279	Incorrect Execution-assigned Permissions	While it is executing, the software sets the permissions of an object in a way that violates the intended permissions that have been specified by the user.
281	Improper Preservation of Permissions	The software does not preserve permissions or incorrectly preserves permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions than intended.
560	Use of umask() with chmod-style Argument	The product calls umask() with an incorrect argument that is specified as if it is an argument to chmod().
732	Incorrect Permission Assignment for Critical Resource	The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

4.14.3 Secondary Cluster: Access Management

This cluster covers various situations related to the management of resource owners and access rights. There is no sufficient white-box content in CWE descriptions for this cluster

This cluster has 5 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

282	Improper Ownership Management	The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource.
283	Unverified Ownership	The software does not properly verify that a critical resource is owned by the proper entity.
284	Access Control (Authorization) Issues	Improper administration of the permissions to the users of a system can result in unintended access to sensitive files.
286	Incorrect User Management	The software does not properly manage a user within its environment.
708	Incorrect Ownership Assignment	The software assigns an owner to a resource, but the owner is outside of the intended control sphere.

4.15 Primary Cluster: Privilege

This cluster of weaknesses relates to code regions with inappropriate privilege level. The common characteristics of this cluster include:

- Privilege level
- Privileged operation
- Region with elevated privilege
- Privilege check
- Resource access operation

Through these characteristics the “Privilege” cluster is associated with the following clusters:

- Access control
- Exception management
- Resource access

There are 12 CWEs in this cluster. Most CWE descriptions do not have sufficient white-box content.

This cluster has only one secondary cluster (no further differentiation).

4.15.1 Secondary Cluster: Privilege

This cluster has 12 CWEs. Only 1 CWEs in the cluster is discernible. 11 CWEs are non-discernible.

4.15.1.1 SFP36 Privilege

Software Fault Pattern - a weakness where the code path has all of the following:

- start statement that performs a privileged operation
- end statement that accesses resource where the access is performed at elevated privilege

Where “access at elevated privilege” is defined by a scenario when the code path does not contain a statement that drops privilege

The following table lists all discernible CWEs that contribute to this software fault pattern:

272	Least Privilege Violation	The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.
------------	---------------------------	--

The following table lists all non-discernible CWEs in this cluster:

9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	If elevated access rights are assigned to EJB methods, then an attacker can take advantage of the permissions to exploit the software system.
250	Execution with Unnecessary Privileges	The software performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.
266	Incorrect Privilege Assignment	A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.
267	Privilege Defined With Unsafe Actions	A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity.
268	Privilege Chaining	Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination.
269	Improper Privilege Management	The software does not properly assign, modify, or track privileges for an actor, creating an unintended sphere of control for that actor.
270	Privilege Context Switching Error	The software does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of control.
271	Privilege Dropping / Lowering Errors	The software does not drop privileges before passing control of a resource to an actor that does not have those privileges.
274	Improper Handling of Insufficient Privileges	The software does not handle or incorrectly handles when it has insufficient privileges to perform an operation, leading to resultant weaknesses.

520	.NET Misconfiguration: Use of Impersonation	Allowing a .NET application to run at potentially escalated levels of access to the underlying operating and file systems can be dangerous and result in various forms of attacks.
653	Insufficient Compartmentalization	The product does not sufficiently compartmentalize functionality or processes that require different privilege levels, rights, or permissions.

4.16 Primary Cluster: Channel

These cluster groups weaknesses related to various “protocol” issues. The common characteristics of this cluster include:

- Channel
- Computation
- Component entry point

Through these characteristics the “Channel” cluster is associated to the following clusters:

- Entry point
- Authentication
- Information leak
- Tainted input
- Exception management

There are 13 CWEs in this cluster. All CWEs in this cluster are non-discernible, as protocol characteristics are derived properties of the computation. Each actor participating in a protocol performs according to a certain role defined by the protocol. Deviation from the correct computation flow defined by the protocol are usually non discernible without some other representation of the correct protocol, and even then the full property check is often an intractable problem. Additional research needs to be performed to discover white-box patterns associated with protocol issues.

The Channel cluster includes the following 2 secondary clusters:

- Channel attack – this cluster covers various attack patterns related to channels.

- Protocol error – this cluster covers various deviations between the required protocol and its implementation by the actors

4.16.1 Secondary Cluster: Channel Attack

This cluster covers various attack patterns related to channels.

This cluster has 8 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

290	Authentication Bypass by Spoofing	This attack-focused weakness is caused by improperly implemented authentication schemes that are subject to spoofing attacks.
294	Authentication Bypass by Capture-replay	A capture-replay flaw exists when the design of the software makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).
300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	The product does not adequately verify the identity of actors at both ends of a communication channel, or does not adequately ensure the integrity of the channel, in a way that allows the channel to be accessed or influenced by an actor that is not an endpoint.
301	Reflection Attack in an Authentication Protocol	Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.
419	Unprotected Primary Channel	The software uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.
420	Unprotected Alternate Channel	The software protects a primary channel, but it does not use the same level of protection for an alternate channel.
421	Race Condition During Access to Alternate Channel	The product opens an alternate channel to communicate with an authorized user, but the channel is accessible to other actors.
441	Unintended Proxy/Intermediary	A product can be used as an intermediary or proxy between an attacker and the ultimate target, so that the attacker can either bypass access controls or hide activities.

4.16.2 Secondary Cluster: Protocol Error

This cluster covers various deviations between the required protocol and its implementation by the actors.

This cluster has 5 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

353	Failure to Add Integrity Check Value	If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission.
435	Interaction Error	An interaction error occurs when two entities work correctly when running independently, but they interact in unexpected ways when they are run together.
436	Interpretation Conflict	Product A handles inputs or steps differently than Product B, which causes A to perform incorrect actions based on its perception of B's state.
437	Incomplete Model of Endpoint Features	A product acts as an intermediary or monitor between two or more endpoints, but it does not have a complete model of an endpoint's features, behaviors, or state, potentially causing the product to perform incorrect actions based on this incomplete model.
757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.

4.17 Primary Cluster: Cryptography

This cluster of weaknesses relates to use of ciphers, keys and other cryptography issues. The common characteristics of this cluster include:

- Sensitive data

- Predictability

Through these characteristics this cluster is associated to the following clusters:

- Authentication
- Information leak
- Predictability

There are 13 CWEs in this cluster. All CWEs are described using non-discernible properties.

The Cryptography cluster includes the following 2 secondary clusters:

- Broken cryptography
- Weak cryptography

4.17.1 Secondary Cluster: Broken Cryptography

This cluster has 5 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

325	Missing Required Cryptographic Step	The software does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than advertised by that algorithm.
327	Use of a Broken or Risky Cryptographic Algorithm	The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.
328	Reversible One-Way Hash	The product uses a hashing algorithm that produces a hash value that can be used to determine the original input, or to find an input that can produce the same hash, more efficiently than brute force techniques.
759	Use of a One-Way Hash without a Salt	The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the input.
760	Use of a One-Way Hash with a Predictable Salt	The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software uses a predictable salt as part of the input.

4.17.2 Secondary Cluster: Weak Cryptography

This cluster has 8 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

261	Weak Cryptography for Passwords	Obscuring a password with a trivial encoding does not protect the password.
322	Key Exchange without Entity Authentication	The software performs a key exchange with an actor without verifying the identity of that actor.
323	Reusing a Nonce, Key Pair in Encryption	Nonces should be used for the present occasion and only once.
324	Use of a Key Past its Expiration Date	The product uses a cryptographic key or password past its expiration date, which diminishes its safety significantly by increasing the timing window for cracking attacks against that key.
326	Weak Encryption	Insufficiently strong encryption schemes may not adequately secure secret data from attackers. Attackers can guess or use brute force attacks to break weakly encrypted schemes.
329	Not Using a Random IV with CBC Mode	Not using a random initialization Vector (IV) with Cipher Block Chaining (CBC) Mode causes algorithms to be susceptible to dictionary attacks.
347	Improperly Verification of Cryptographic Signature	The software does not verify, or incorrectly verifies, the cryptographic signature for data.
640	Weak Password Recovery Mechanism for Forgotten Password	The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak.

4.18 Primary Cluster: Malware

This cluster of weaknesses relates to any malicious code that is present in the software system. Malicious computations are performed as part of the production computation, and may use covert channels as well as regular channels. The common characteristics of this cluster include the following:

- Channel
- Production computation

- Malicious computation
- Covert channel

There are 11 CWEs in this cluster. All CWE are describe using non-discernible properties, as “maliciousness” property is a highly derived property related to the meaning of the computation, rather than to the particular white-box patterns.

The Malware cluster includes the following 2 secondary clusters:

- Malicious code– this cluster covers various scenarios of implanted code with malicious intent
- Covert channel – this cluster covers scenarios which involve covert means of communication that are often used by malicious code

4.18.1 Secondary Cluster: Malicious Code

This cluster covers various scenarios of implanted code with malicious intent.

This cluster has 8 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

69	Failure to Handle Windows ::DATA Alternate Data Stream	The software does not properly prevent access to, or detect usage of, alternate data streams (ADS).
506	Embedded Malicious Code	The application contains code that appears to be malicious in nature.
507	Trojan Horse	The software appears to contain benign or useful functionality, but it also contains code that is hidden from normal operation that violates the intended security policy of the user or the system administrator.
508	Non-Replicating Malicious Code	Non-replicating malicious code only resides on the target system or software that is attacked; it does not attempt to spread to other systems.
509	Replicating Malicious Code (Virus or Worm)	Replicating malicious code, including viruses and worms, will attempt to attack other systems once it has successfully compromised the target system or software.
510	Trapdoor	A trapdoor is a hidden piece of code that responds

		to a special input, allowing its user access to resources without passing through the normal security enforcement mechanism.
511	Logic/Time Bomb	The software contains code that is designed to disrupt the legitimate operation of the software (or its environment) when a certain time passes, or when a certain logical condition is met.
512	Spyware	The software collects personally identifiable information about a human user or the user's activities, but the software accesses this information using other resources besides itself, and it does not require that user's explicit approval or direct input into the software.

4.18.2 Secondary Cluster: Covert Channel

This cluster covers scenarios which involve covert means of communication that are often used by malicious code.

This cluster has 3 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

385	Covert Timing Channel	Covert timing channels convey information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information.
514	Covert Channel	A covert channel is a path used to transfer information in a way not intended by the system's designers.
515	Covert Storage Channel	A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another. What distinguishes this case from that of ordinary operation is that the bits are used to convey encoded information.

4.19 Primary Cluster: Predictability

This cluster groups weaknesses related to random number generators and their properties.

This cluster is related to the following clusters:

- Cryptography

4.19.1 Secondary Cluster: Predictability

There are 15 CWEs in this cluster. All CWEs are described using non-discernible properties.

This cluster has only one secondary cluster (no further differentiation).

This cluster has 15 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

330	Use of Insufficiently Random Values	The software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers.
331	Insufficient Entropy	The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.
332	Insufficient Entropy in PRNG	The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.
333	Improper Handling of Insufficient Entropy in TRNG	True random number generators (TRNG) generally have a limited source of entropy and therefore can fail or block.
334	Small Space of Random Values	The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.
335	PRNG Seed Error	A Pseudo-Random Number Generator (PRNG) uses seeds incorrectly.
336	Same Seed in PRNG	A PRNG uses the same seed each time the product is initialized. If an attacker can guess (or knows) the seed, then he/she may be able to determine the "random" number produced from the PRNG.
337	Predictable Seed in PRNG	A PRNG is initialized from a predictable seed, e.g. using process ID or system time.
338	Use of Cryptographically Weak PRNG	The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG is not cryptographically strong.

339	Small Seed Space in PRNG	A PRNG uses a relatively small space of seeds.
340	Predictability Problems	Weaknesses in this category are related to schemes that generate numbers or identifiers that are more predictable than required by the application.
341	Predictable from Observable State	A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, etc.
342	Predictable Exact Value from Previous Values	An exact value or random number can be precisely predicted by observing previous values.
343	Predictable Value Range from Previous Values	The software's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.
344	Use of Invariant Value in Dynamically Changing Context	The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

4.20 Primary Cluster: UI

This cluster of weaknesses relate to security issues of User Interfaces (UI). The common characteristics of this cluster include:

- Production computation
- Authentication
- Entry point

Through these characteristics this cluster is associated to the following clusters:

- Entry points
- Authentication
- Exception management

There are 14 CWEs in this cluster. All CWEs have insufficient white-box content.

The “UI” cluster includes the following 3 secondary clusters:

- Feature
- Information loss

- Security

4.20.1 Secondary Cluster: Feature

This cluster has 7 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

447	Unimplemented or Unsupported Feature in UI	A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented.
448	Obsolete Feature in UI	A UI function is obsolete and the product does not warn the user.
449	The UI Performs the Wrong Action	The UI performs the wrong action with respect to the user's request.
450	Multiple Interpretations of UI Input	The UI has multiple interpretations of user input but does not prompt the user when it selects the less secure interpretation.
451	UI Misrepresentation of Critical Information	The UI does not properly represent critical information to the user, allowing the information - or its source - to be obscured or spoofed. This is often a component in phishing attacks.
549	Missing Password Field Masking	The software fails to mask passwords during entry, increasing the potential for attackers to observe and capture passwords.
655	Insufficient Psychological Acceptability	The software has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose.

4.20.2 Secondary Cluster: Information Loss

This cluster has 4 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

221	Information Loss or Omission	The software does not record, or improperly records, security-relevant information that leads to an incorrect decision or hampers later analysis.
------------	------------------------------	---

222	Truncation of Security-relevant Information	The application truncates the display, recording, or processing of security-relevant information in a way that can obscure the source or nature of an attack.
223	Omission of Security-relevant Information	The application does not record or display information that would be important for identifying the source or nature of an attack, or determining if an action is safe.
224	Obscured Security-relevant Information by Alternate Name	The software records security-relevant information according to an alternate name of the affected entity, instead of the canonical name.

4.20.3 Secondary Cluster: Security

This cluster has 3 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

356	Product UI does not Warn User of Unsafe Actions	The software's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system.
357	Insufficient UI Warning of Dangerous Operations	The user interface provides a warning to a user regarding dangerous or sensitive operations, but the warning is not noticeable enough to warrant attention.
446	UI Discrepancy for Security Feature	The user interface does not correctly enable or configure a security feature, but the interface provides feedback that causes the user to believe that the feature is in a secure state.

4.21 Primary Cluster: Other

This cluster of weaknesses relates to miscellaneous architecture, design and implementation issues. The common characteristics of this cluster include:

- Production computation
- Exception management
- Computation building blocks

Through these characteristics this cluster is associated to the following clusters:

- Risky computation patterns
- Exception management

There are 45 CWEs in this cluster. All CWEs have insufficient white-box content.

The Other cluster includes the following 5 secondary clusters:

- Architecture
- Design
- Implementation
- Compiler
- Suspicious syntactic constructs– this cluster covers several suspicious patterns that may contribute to other weaknesses and are often indicators of poor code quality

4.21.1 Secondary Cluster: Architecture

This cluster has 11 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

348	Use of Less Trusted Source	The software has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.
359	Privacy Violation	Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.
602	Client-Side Enforcement of Server-Side Security	The software has a server that relies on the client to implement a mechanism that is intended to protect the server.
637	Failure to Use Economy of Mechanism	The software uses a more complex mechanism than necessary, which could lead to resultant weaknesses when the mechanism is not correctly understood, modeled, configured, implemented, or used.

649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	The software uses obfuscation or encryption of inputs that should not be mutable by an external actor, but the software does not use integrity checks to detect if those inputs have been modified.
654	Reliance on a Single Factor in a Security Decision	A protection mechanism relies exclusively, or to a large extent, on the evaluation of a single condition or the integrity of a single object or entity in order to make a decision about granting access to restricted resources or functionality.
656	Reliance on Security through Obscurity	The software uses a protection mechanism whose strength depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to defeat the mechanism.
657	Violation of Secure Design Principles	The product violates well-established principles for secure design.
671	Lack of Administrator Control over Security	The product uses security features in a way that prevents the product's administrator from tailoring security settings to reflect the environment in which the product is being used. This introduces resultant weaknesses or prevents it from operating at a level of security that is desired by the administrator.
693	Protection Mechanism Failure	The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.
749	Exposed Dangerous Method or Function	The software provides an Applications Programming Interface (API) or similar interface for interaction with external actors, but the interface includes a dangerous method or function that is not properly restricted.

4.21.2 Secondary Cluster: Design

This cluster has 28 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

115	Misinterpretation of Input	The software misinterprets an input, whether from an attacker or another product, in a security-
------------	----------------------------	--

		relevant fashion.
187	Partial Comparison	The software performs a comparison that only examines a portion of a factor before determining whether there is a match, such as a substring, leading to resultant weaknesses.
188	Reliance on Data/Memory Layout	The software makes invalid assumptions about how protocol data or memory is organized at a lower level, resulting in unintended program behavior.
193	Off-by-one Error	A product calculates or uses an incorrect maximum or minimum value that is 1 more, or 1 less, than the correct value.
349	Acceptance of Extraneous Untrusted Data With Trusted Data	The software, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.
405	Asymmetric Resource Consumption (Amplification)	Software that fails to appropriately monitor or control resource consumption can lead to adverse system performance.
406	Insufficient Control of Network Message Volume (Network Amplification)	The software does not sufficiently monitor or control transmitted network traffic volume, so that an actor can cause the software to transmit more traffic than should be allowed for that actor.
407	Algorithmic Complexity	An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.
408	Incorrect Behavior Order: Early Amplification	The software allows an entity to perform a legitimate but expensive operation before authentication or authorization has taken place.
409	Improper Handling of Highly Compressed Data (Data Amplification)	The software does not handle or incorrectly handles a compressed input with a very high compression ratio that produces a large output.
410	Insufficient Resource Pool	The software's resource pool is not large enough to handle peak demand, which allows an attacker to prevent others from accessing the resource by using a (relatively) large number of requests for resources.

430	Deployment of Wrong Handler	The wrong "handler" is assigned to process an object.
462	Duplicate Key in Associative List (Alist)	Duplicate keys in associative lists can lead to non-unique keys being mistaken for an error.
463	Deletion of Data Structure Sentinel	The accidental deletion of a data-structure sentinel can cause serious programming logic problems.
464	Addition of Data Structure Sentinel	The accidental addition of a data-structure sentinel can cause serious programming logic problems.
480	Use of Incorrect Operator	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
581	Object Model Violation: Just One of Equals and Hashcode Defined	The software fails to maintain equal hashcodes for equal objects.
595	Comparison of Object References Instead of Object Contents	The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.
596	Incorrect Semantic Object Comparison	The software does not correctly compare two objects based on their conceptual content.
618	Exposed Unsafe ActiveX Method	An ActiveX control is intended for use in a web browser, but it exposes dangerous methods that perform actions that are outside of the browser's Best Practice (e.g. the zone or domain).
648	Incorrect Use of Privileged APIs	The application does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.
670	Always-Incorrect Control Flow Implementation	The code contains a control flow path that does not reflect the algorithm that the path is intended to implement, leading to incorrect behavior any time this path is navigated.
682	Incorrect Calculation	The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
691	Insufficient Control Flow Management	The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.

696	Incorrect Behavior Order	The software performs multiple related behaviors, but the behaviors are performed in the wrong order in ways which may produce resultant weaknesses.
697	Insufficient Comparison	The software compares two entities in a security-relevant context, but the comparison is insufficient, which may lead to resultant weaknesses.
698	Redirect Without Exit	The web application sends a redirect to another location, but instead of exiting, it executes additional code.
705	Incorrect Control Flow Scoping	The software does not properly return control flow to the proper location after it has completed a task or detected an unusual condition.
483	Incorrect Block Delimitation	The code does not explicitly delimit a block that is intended to contain 2 or more statements, creating a logic error.

4.21.3 Secondary Cluster: Implementation

This cluster has 5 CWEs. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

216	Containment Errors (Container Errors)	This tries to cover various problems in which improper data are included within a "container."
358	Improperly Implemented Security Check for Standard	The software does not implement or incorrectly implements one or more security-relevant checks as specified by the design of a standardized algorithm, protocol, or technique.
398	Indicator of Poor Code Quality	The code has features that do not directly introduce a weakness or vulnerability, but indicate that the product has not been carefully developed or maintained.
623	Unsafe ActiveX Control Marked Safe For Scripting	An ActiveX control is intended for restricted use, but it has been marked as safe-for-scripting.
710	Coding Standards Violation	The software does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

4.21.4 Secondary Cluster: Compiler

This cluster has 1 CWE. All CWEs in the cluster are non-discernible.

The following table lists all non-discernible CWEs in this cluster:

733	Compiler Optimization Removal or Modification of Security-critical Code	The developer builds a security-critical protection mechanism into the software but the compiler optimizes the program such that the mechanism is removed or modified.
------------	---	--

APPENDIX A: Software Fault Patterns

The following table (Table 8) enumerates Software Fault Patterns (SFPs). For each, it lists Primary Cluster that belongs to and definition.

Table 8. Software Fault Patterns

Cluster	SFP ID & Name	Software Fault Pattern Definition
Risky Values	SFP1 - Glitch In Computation	Software Fault Pattern – a weakness where the code path has all of the following: <ul style="list-style-type: none">○ an end statement that performs an identifiable operation on data producing some actual value of a datatype and○ the data is inappropriate to the operation resulting in the value that is unexpected for the datatype and the operation
Unused Entities	SFP2 - Unused entities	Software Fault Pattern – an entity that does not have incoming usage relationships
API	SFP3 - Use of an improper API	Software Fault Pattern - a weakness where the code path has all of the following: <ul style="list-style-type: none">○ an end statement that performs an API call where the call is not appropriate for the given platform

Cluster	SFP ID & Name	Software Fault Pattern Definition
Exception Management	SFP4 - Unchecked status condition	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that produces a status condition ○ an end statement that incorrectly acts on the status condition ○ where “incorrect act” is defined as exactly one of the following: <ul style="list-style-type: none"> ○ status condition never obtained and used ○ status condition obtained but not used ○ status incorrectly validated such that actual and expected status mismatch

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP5 - Ambiguous Exception type	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that requires exception signature where the exception signature is more general than the corresponding exception profile <p>Where:</p> <ul style="list-style-type: none"> ○ Exception profile is the set of exceptions thrown by a code fragment $EP=\{e1, ..., ek\}$ ○ Exception signature is the set of exceptions declared for the try-block (in which case it should match the exception profile of the try-block) or at the method declaration (in which case it should match the exception profile of the entire method) $ES=\{s1,...,sl\}$ ○ Exception signature (ES) is more general than the exception profile (EP) of the corresponding code fragment if ES contains s which is a supertype of one or more ei in EP
	SFP6 - Incorrect Exception Behavior	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that assigns incorrect value to status condition ○ an end statement that uses incorrect value of status condition

Cluster	SFP ID & Name	Software Fault Pattern Definition
Memory Access	SFP7 - Faulty pointer use	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs use of pointer with NULL value or “out of range” value <p>Where a “out of range value ” is defined as access to memory chunk through exactly one of the following:</p> <ul style="list-style-type: none"> ○ faulty address obtained as a subtraction of two pointers to different memory chunks or ○ faulty type such as use of a pointer to access a structure element where the pointer was cast from a data item that is not of a structure datatype

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP8 - Faulty buffer access	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs a Buffer Access Operation and where exactly one of the following is true: <ul style="list-style-type: none"> ○ the access position of the Buffer access Operation is outside of the buffer or ○ the access position of the Buffer access Operation is inside the buffer and the size of the buffer is greater than the actual size of the bufferdata being accessed is greater than the remaining size of the buffer at the access position <p>Where the Buffer Access Operation is a statement that performs access to a data item of a certain size at access position. The access position of a Buffer access Operation is related to a certain buffer and can be either inside the buffer or outside the buffer.</p>
	SFP9 - Faulty string expansion	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that allocates a buffer ○ an end statement that performs an implicit buffer access through function call that is characterized by buffer parameters such as the buffer size and the expected buffer size where the expected buffer size is greater than the actual buffer size

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP10 - Incorrect buffer length computation	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs memory allocation for a datatype based on an existing data item of datatype where the computed length of the buffer is incorrect <p>Where incorrect length of the buffer involves exactly one of the following:</p> <ul style="list-style-type: none"> ○ size of requested buffer is smaller than needs to be ○ size of requested buffer is bigger than needs to be
	SFP11 - Improper NULL termination	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that passes a data item to a null-terminated string operation where the data item is non-null-terminated <p>Where the data can become non-null-terminated in at least one of the following ways:</p> <ul style="list-style-type: none"> ○ data originated from a length-based string operation where the terminator is not automatically added ○ null-terminated string was incorrectly transferred and the terminator was omitted ○ the null terminator has been overwritten ○ array is interpreted as a string where the null terminator is not present in the array

Cluster	SFP ID & Name	Software Fault Pattern Definition
Memory Management	SFP12 - Faulty memory release	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that releases memory via a reference where the reference points to either incorrect address or incorrect address type
Resource Management	SFP13 - Unrestricted consumption	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs resource allocation where there is a loopback path and the resource is not released and the consumption limit is absent <p>Where “allocation control” is defined as the condition associated with the code path that limits the number of the allocated resource instances.</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP14 - Failure to release resource	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement performs resource allocation ○ an end statement that loses identity of the resource and the resource is not in released state <p>Where “loses identity” is defined as one of the following:</p> <ul style="list-style-type: none"> ○ resource identity has not been not stored when received ○ resource identity has been obtained but was over-written (missing beyond recovery) ○ resource identity was never passed to the resource release function ○ resource identity is stored in a data item and the data item goes out of scope (no more aliases remain) ○ resource identity is stored in a data item and the data item is destroyed
	SFP15 - Faulty resource use	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that performs release of a resource ○ an end statement that performs access to a resource and the resource is in released state

Cluster	SFP ID & Name	Software Fault Pattern Definition
Path Resolution	SFP16 - Path traversal	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that accepts input ○ an end statement that opens a file using a file path (consisting of a directory name and a filename) where the input is part of the file path and the file path is insecure <p>Where “insecure file path” is defined as the path of resources that are at least one of the following:</p> <ul style="list-style-type: none"> • Resources outside of the access root • Set of security-sensitive resources
	SFP17 - Failed chroot jail	<p>Software Fault Pattern – a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that has at least one of the following: <ul style="list-style-type: none"> ○ performs a chroot or ○ performs a chdir, ○ an end statement that opens a file where chroot is activated and the current working directory is outside of the chroot jail

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP18 - Link in resource name resolution	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that accepts input ○ an end statement that opens a file using a file path where the file path is not link-sanitized ○ where “not link-sanitized” is defined as exactly one of the following: <ul style="list-style-type: none"> • Check for link not performed • Check for link does not cover every segment in the file path
Synchronization	SFP19 - Missing lock	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that accesses a shared entity and the entity is improperly synchronized <p>Where shared entity is one of the follows:</p> <ul style="list-style-type: none"> ○ resource of the particular resource type ○ shared data (including static variables) of a particular datatype <p>Where the “improper synchronization” is defined as the situation where there does not exist any lock that synchronizes the shared entity along the given code path or when locks are not adequate.</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP20 - Race condition window	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that checks the status of a resource ○ an end statement that performs access to the same resource where the second resource access occurs on the conditional branch of start statement and the start statement is not atomic
	SFP21 - Multiple locks/unlocks	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that performs call to change resource's locking state ○ an end statement that performs call for the same locking state that the resource is already in
	SFP22 - Unrestricted lock	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs lock of a resource and the resource is externally accessible and there is no alternative flow (the flow will be stuck if the resource becomes locked externally)

Cluster	SFP ID & Name	Software Fault Pattern Definition
Information Leak	SFP23 - Exposed data	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement performs moving data where the data is sensitive and the data is inadequately protected <p>Where “inadequately protected data” is defined as exactly one of the following:</p> <ul style="list-style-type: none"> • data that is not encrypted ((in cleartext, in plaintext) • data that is not sanitized <p>Where “sensitive data” is defined as used in APIs that are intended for handling sensitive data (for example passwords).</p>
Tainted Input	SFP24 - Tainted Input to Command	<p>Software Fault Pattern - A weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that accepts input data ○ an end statement that executes destination command where the input data is part of destination command and the input data is undesirable <p>Where “input is undesirable” is defined through the following scenarios:</p> <ul style="list-style-type: none"> ○ not validated ○ incorrectly validated against special characters and symbols that trigger certain functionality during execution of destination command (discernible) and against applicable design specification

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP25 - Tainted input to variable	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ the end statement that uses tainted data value <p>Where tainted data value use is defined as externally controlled value obtained through at least one of the following scenarios:</p> <ul style="list-style-type: none"> ○ not validated user input ○ not validated configuration settings ○ not validated environment variables <p>Where tainted data value use is defined by exactly one of the following scenarios:</p> <ul style="list-style-type: none"> -- used as a program's control variable -- assigned to a variable <p>Where "user input" is defined as data originating through a certain platform-specific resource which is accessed using a system call.</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP26 - Composite tainted input	<p>Software Fault Pattern – is a minimal code path which has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that accepts input data through exactly one of the following: <ul style="list-style-type: none"> ○ executing “input command” or ○ component entry point (API that can be invoked by platform) ○ an end statement that is an end statement of another SFP that uses a data value where the input data contributes to the data value ○ the condition of the SFP of the end statement is satisfied
	SFP27 - Tainted Input to Environment	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ the end statement that calls a tainted control element <p>Where tainted control element is defined by exactly one of the following scenarios:</p> <ul style="list-style-type: none"> ○ loading library from untrusted source ○ dynamically loading code from untrusted source ○ external function hook

Cluster	SFP ID & Name	Software Fault Pattern Definition
Entry Points	SFP28 - Unexpected access points	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ statement that defines an entry point into an application where the entry point is not required by production code <p>Where “entry point that is not required by production code” is defined as any functionality not required by the production code such as either:</p> <ul style="list-style-type: none"> ○ debug code, or ○ test code or ○ access to data

Cluster	SFP ID & Name	Software Fault Pattern Definition
Authentication	SFP29 - Faulty endpoint authentication	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that performs input with the input data ○ an end statement that performs condition check that involves data value where the data value is a property of the input data and the property provides insufficient endpoint authentication <p>Where “property of the input data” is defined as the result of the API calls marked in the knowledge base as returning properties of the input data, where the input data is passed as the key parameter to the call. There can be chains of such properties, for example getRemoteAddr()->getByName()->getCanonicalHostName(), where the first call operates on a request and the last call return a property of that request.</p> <p>Where “property provides insufficient endpoint authentication” is defined a property that is marked in the knowledge base as insufficient for authentication.</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP30 - Missing endpoint authentication	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that performs input with input data ○ an end statement that accesses the input data where the end statement occurs at the region that is not guarded by endpoint authentication conditions of the input data <p>Where “endpoint authentication condition of an input data” is a condition that involves data value that is a property of the input data.</p> <p>Where “statement access input data” is defined as an API call which is defined in the knowledge base as accessing the input data, and where the input data is a passed as the key parameter to the call.</p> <p>Where a “region guarded by condition” is a code fragment that starts at the selected branch of the condition (the branch where the condition is considered validated).</p>
	SFP31 - Missing authentication	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs a critical operation where the end statement occurs at the region that is not guarded by authentication conditions <p>Where “authentication condition” is a condition that involves a data value that is a result of an API call that is marked in the knowledge base as authentication call.</p> <p>Where “statement performs critical operation” is defined as an API call which is defined in the knowledge base as performing a critical operation (for example, a password change).</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
	SFP32 - Multiple binds to the same port	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs binding of a socket where the socket allows multiple bindings <p>Where “socket allows multiple bindings” is defined as a specific value “bind to all interfaces” INADD_ANY or 0.0.0.0</p>
	SFP33 - Hardcoded sensitive data	<p>Software Fault Pattern – a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ a start statement that accepts sensitive data item ○ an end statement that performs comparison where one operand is the sensitive data item and the other operand is a constant value
	SFP34 - Unrestricted authentication	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ an end statement that performs authentication where the unvalidated branch has a loopback path and the authentication control is absent <p>Where “authentication control” is defined as the condition associated with the code path that limits the number of the authentication attempts.</p> <p>Where a “statement that performs authentication” is an authentication condition.</p>

Cluster	SFP ID & Name	Software Fault Pattern Definition
Access Control	SFP35 - Insecure resource access	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ End statement that performs a resource access to a resource where the end statement occurs at the region that is not guarded by access control conditions involving the resource <p>Where “access control condition involving a resource” is a condition that involves a data value that is a result of an API call that is marked in the knowledge base as authentication call, where the API call is passed another data value that is a property of the resource.</p> <p>Where “statement performs resource access” is defined as an API call which is defined in the knowledge base as performing a resource access.</p> <p>Where “property of resource” is defined as the result of the API calls marked in the knowledge base as returning properties of the resource, where the resource identity is passed as the key parameter to the call. There can be chains of such properties.</p>
Privilege	SFP36 - Privilege	<p>Software Fault Pattern - a weakness where the code path has all of the following:</p> <ul style="list-style-type: none"> ○ start statement that performs a privileged operation ○ end statement that accesses resource where the access is performed at elevated privilege <p>Where “access at elevated privilege” is defined by a scenario when the code path does not contain a statement that drops privilege.</p>

APPENDIX B: Software Fault Patterns and Corresponding Impacts

Refer to Table 9 for SFP and their corresponding impacts.

Table 9. Software Fault Patterns with Corresponding Impacts

SFP #	Primary Cluster	SFP Description	Impact
SFP1	Risky values	Glitch in Computation	Loss of integrity of data in use; loss of integrity of service; contributes to other SFP (condition, buffer properties, etc.); loss of availability of service (e.g., divide by zero)
SFP2	Unused Entities	Unused entities	Indicator of poor quality
SFP3	API	Use of an Improper API	Impact is described in the knowledge base; API call is a foot-hold
SFP4	Exception Management	Unchecked status condition	Loss of Integrity of service; contributes to other SFP
SFP5	Exception Management	Ambiguous exception type	Loss of Integrity of service; contributes to other SFP
SFP6	Exception Management	Incorrect Exception Behavior	Loss of Integrity of service; loss of availability of service; loss of confidentiality
SFP7	Memory Access	Faulty pointer use	Loss of availability of service (may crash); contributes to SFP4

SFP #	Primary Cluster	SFP Description	Impact
SFP8	Memory Access	Faulty buffer access	Loss of availability of service (write access); subversion of service (especially "bulk" write access); loss of integrity of service (write access); loss of confidentiality (read access)
SFP9	Memory Access	Faulty string expansion	Loss of availability of service (may crash); loss of integrity of service (may produce incomplete results); subversion of service
SFP10	Memory Access	Incorrect bufer length calculation	Loss of Integrity of data in use; loss of Integrity of service; contributes to SFP8; contributes to SFP9
SFP11	Memory Access	Improper NULL termination	Loss of availability of service (for operations that write to the buffer); loss of integrity of service (may corrupt data); subversion of service
SFP12	Memory Management	Faulty memory release	Loss of availability of resource (resource is not released as intended); shutdown of service (may crash); loss of availability of service (lockdown of service when lock resource is involved)
SFP13	Resource Management	Unrestricted consumption	Loss of availability of resource; loss of availability of service
SFP14	Resource Management	Failure to release resource	Loss of availability of resource; loss of availability of service
SFP15	Resource Management	Faulty resource use	Loss of integrity of service; loss of availability of service; loss of availability of resource

SFP #	Primary Cluster	SFP Description	Impact
SFP16	Path resolution	Path traversal	Loss of confidentiality; loss of integrity of data at rest
SFP17	Path resolution	Failed chroot jail	contributes to SFP16
SFP18	Path resolution	Link in resource name resolution	contributes to SFP16
SFP19	Synchronization	Missing lock	Loss of availability of resource; loss of availability of service; loss of integrity of data; loss of integrity of service; contributes to other SFP
SFP20	Synchronization	Race condition window	Loss of availability of service; loss of availability of resource; loss of integrity of service; contributes to other SFP
SFP21	Synchronization	Multiple locks/unlocks	Loss of availability of service; loss of availability of resource; loss of integrity of service; contributes to other SFP
SFP22	Synchronization	Unrestricted lock	Loss of availability of service; loss of availability of resource; loss of integrity of service
SFP23	Information leak	Exposed data	Loss of Confidentiality of data
SFP24	Tainted Input	Tainted data to command	subversion of service (for "code modification" destination commands); loss of integrity of data (for destination commands that modify data)

SFP #	Primary Cluster	SFP Description	Impact
SFP25	Tainted Input	Tainted input to variable	Loss of integrity of service (distortion, shutdown or lock when control variables are modified); loss of integrity of data
SFP26	Tainted Input	Composite tainted data	no impact of its own - this is a way to describe complex weaknesses
SFP27	Tainted Input	Tainted input to environment	Loss of integrity of service; loss of integrity of data
SFP28	Entry points	Unexpected entry point	Loss of confidentiality; contributes to other SFPs
SFP29	Authentication	Faulty endpoint authentication	Loss of confidentiality; contributes to other SFPs
SFP30	Authentication	Missing endpoint authentication	Loss of confidentiality; contributes to other SFPs
SFP31	Authentication	Missing authentication	Loss of confidentiality; contributes to other SFPs
SFP32	Authentication	Multiple binds to the same port	Loss of confidentiality; contributes to other SFPs
SFP33	Authentication	Hardcoded sensitive data	Loss of confidentiality
SFP34	Authentication	Unrestricted authentication	Loss of availability of service
SFP35	Access control	Insecure resource access	Loss of confidentiality; loss of integrity of data;
SFP36	Privilege	Privilege	Contributes to other SFPs

List of Acronyms, Abbreviations, and Symbols

ACRONYM	DEFINITION
AFRL	Air Force Research Laboratory
API	Application Programming Interface
CIO	Chief Information Officer
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DoD	Department of Defense
EISTS	Embedded Information Systems Technology Support
KDM	Knowledge Discovery Metamodel (OMG Specification)
NII	Network and Information Integration
NIST	National Institute of Standards and Technology
OASD	Office of the Assistant Secretary of Defense
OMG	Object Management Group Consortium
SBVR	Semantic Business Vocabulary and Rules (OMG Spec.)
SFP	Software Fault Pattern
SOW	Supplier Statement of Work
SwA	Software Assurance
TCG	Test Case Generator
TIC	Tainted Input to Command
TIE	Tainted Input to Environment
TIV	Tainted Input to Variable
VPAD	Vulnerability Path Analysis and Demonstration
WK	Weakness Kernel
XMI	XML Metadata Interchange



DEPARTMENT OF THE AIR FORCE
AIR FORCE RESEARCH LABORATORY
WRIGHT-PATTERSON AIR FORCE BASE OHIO 45433

21 March 2013

MEMORANDUM FOR: Defense Technical Information Center/OCA
8725 John J. Kingman Rd, Suite 0944
Ft Belvoir, VA 22060-6218

FROM: RY STINFO Tech Editing Office
AFRL/RYOX (STINFO)
2241 Avionics Circle
Bldg 620, Room 2AL48
Wright-Patterson AFB, OH 45433-7320

SUBJECT: Notice of Change for Technical Report

1. Reference: (U) Embedded Information Systems Technology Support (EISTS). Task Order 0006: Vulnerability Path Analysis and Demonstration (VPAD). Volume 2 - White Box Definitions of Software Fault Patterns, ADB381215.
 - Change distribution statement from C to A.
 - PAO Case Number: 88ABW-2013-1292, cleared March 15, 2013.
2. Please call me at DSN 798-8489 if more information is needed.
3. Thank you for your attention to this matter.

MITCHELL.LOLITA.V.1262526573
Digitally signed by
MITCHELL.LOLITA.V.1262526573
DN: c=US, o=U.S. Government, ou=DoD, ou=PKI,
ou=USAF, cn=MITCHELL.LOLITA.V.1262526573
Date: 2013.03.20 15:20:36 -04'00'

LOLITA V. MITCHELL
STINFO Officer